MachineLearning

1st Assignment - Shahid Beheshti University February 26, 2023

Outline

Packages

In [620]:

```
import numpy as np
import pandas as pd
import random
#plotting packages
import matplotlib.pyplot as plt
import seaborn as sns
#splitting data to train and test
from sklearn.model_selection import train_test_split
from scipy import stats
#Dataset using in exercise 5 and exercise 6
from sklearn import datasets
from sklearn.datasets import load_diabetes
#feature scaling
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import RobustScaler
#Models
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVR
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
#model selection
from sklearn.model_selection import GridSearchCV
#metrics
import math
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.metrics import precision_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
#ols for forward selection
import statsmodels.api as sm
#wrapper methods
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
#filter methods feature selection
from sklearn import preprocessing
from sklearn.feature_selection import mutual_info_classif
import shap
from scipy.stats import chisquare
from sklearn.feature_selection import r_regression, SelectKBest,SelectPercentile,f_regression
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
#clustering
from sklearn.cluster import KMeans
#multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
#binning
from scipy.stats import binned_statistic
#extra
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
#larger dataframe
pd.set_option('display.max_columns', 500)
pd.set_option('max_colwidth', None)
```

Data Exploration

Checking out the Diabet dataset using in exercise 5 and exercise 6

In [631]:

```
diabetes = datasets.load_diabetes()
```

In [632]:
```
print(diabetes.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_samples` (i.e. the sum of squares of each colum
n totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html (https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html)

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [633]:
```
X,Y = load_diabetes(return_X_y=True)
```

In [634]:
```
X.shape
```

Out[634]: (442, 10)

In [635]:
```
type(X)
```

Out[635]: numpy.ndarray

In [636]:
```
X = pd.DataFrame(X,columns=['age','sex','bmi','bp','tc','ldl','hdl','tch','ltg','glu'])
X
```

24	-0.063635	-0.044642	0.035829	-0.022885	-0.030464	-0.018850	-0.006584	-0.002592	-0.025953	-0.054925
25	-0.067268	0.050680	-0.012673	-0.040099	-0.015328	0.004636	-0.058127	0.034309	0.019196	-0.034215
26	-0.107226	-0.044642	-0.077342	-0.026328	-0.089630	-0.096198	0.026550	-0.076395	-0.042571	-0.005220
27	-0.023677	-0.044642	0.059541	-0.040099	-0.042848	-0.043589	0.011824	-0.039493	-0.015999	0.040343
28	0.052606	-0.044642	-0.021295	-0.074527	-0.040096	-0.037639	-0.006584	-0.039493	-0.000612	-0.054925
29	0.067136	0.050680	-0.006206	0.063187	-0.042848	-0.095885	0.052322	-0.076395	0.059424	0.052770
30	-0.060003	-0.044642	0.044451	-0.019442	-0.009825	-0.007577	0.022869	-0.039493	-0.027129	-0.009362
31	-0.023677	-0.044642	-0.065486	-0.081413	-0.038720	-0.053610	0.059685	-0.076395	-0.037129	-0.042499
32	0.034443	0.050680	0.125287	0.028758	-0.053855	-0.012900	-0.102307	0.108111	0.000272	0.027917
33	0.030811	-0.044642	-0.050396	-0.002228	-0.044223	-0.089935	0.118591	-0.076395	-0.018114	0.003064
34	0.016281	-0.044642	-0.063330	-0.057313	-0.057983	-0.048912	0.008142	-0.039493	-0.059471	-0.067351
35	0.048974	0.050680	-0.030996	-0.049291	0.049341	-0.004132	0.133318	-0.053516	0.021311	0.019633
36	0.012648	-0.044642	0.022895	0.052858	0.008063	-0.028558	0.037595	-0.039493	0.054720	-0.025930

In [528]:
```
Diabet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   tc      442 non-null    float64
 5   ldl     442 non-null    float64
 6   hdl     442 non-null    float64
 7   tch     442 non-null    float64
 8   ltg     442 non-null    float64
 9   glu     442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
```

No null values , and all features are float

In [529]:
```
Diabet.describe()
```

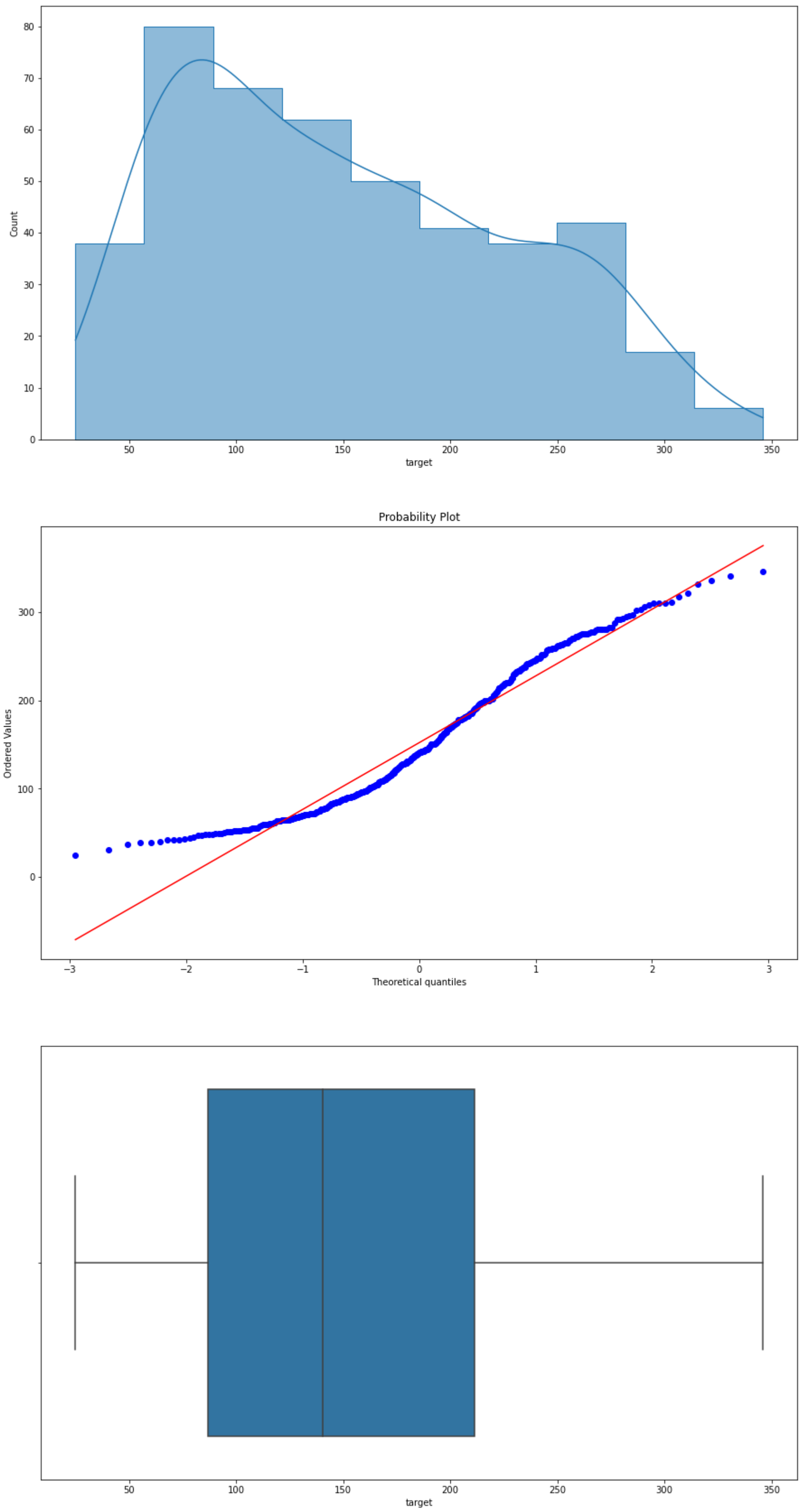
Out[529]:

	age	sex	bmi	bp	tc	ldl	hdl	tch	ltg	glu	target
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	442.000000
mean	-1.444295e-18	2.543215e-18	-2.255925e-16	-4.854086e-17	-1.428596e-17	3.898811e-17	-6.028360e-18	-1.788100e-17	9.243486e-17	1.351770e-17	152.133484
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	77.093005
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123988e-01	-1.267807e-01	-1.156131e-01	-1.023071e-01	-7.639450e-02	-1.260971e-01	-1.377672e-01	25.000000
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665608e-02	-3.424784e-02	-3.035840e-02	-3.511716e-02	-3.949338e-02	-3.324559e-02	-3.317903e-02	87.000000
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670422e-03	-4.320866e-03	-3.819065e-03	-6.584468e-03	-2.592262e-03	-1.947171e-03	-1.077698e-03	140.500000
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564379e-02	2.835801e-02	2.984439e-02	2.931150e-02	3.430886e-02	3.243232e-02	2.791705e-02	211.500000
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320436e-01	1.539137e-01	1.987880e-01	1.811791e-01	1.852344e-01	1.335973e-01	1.356118e-01	346.000000

visualizing target values

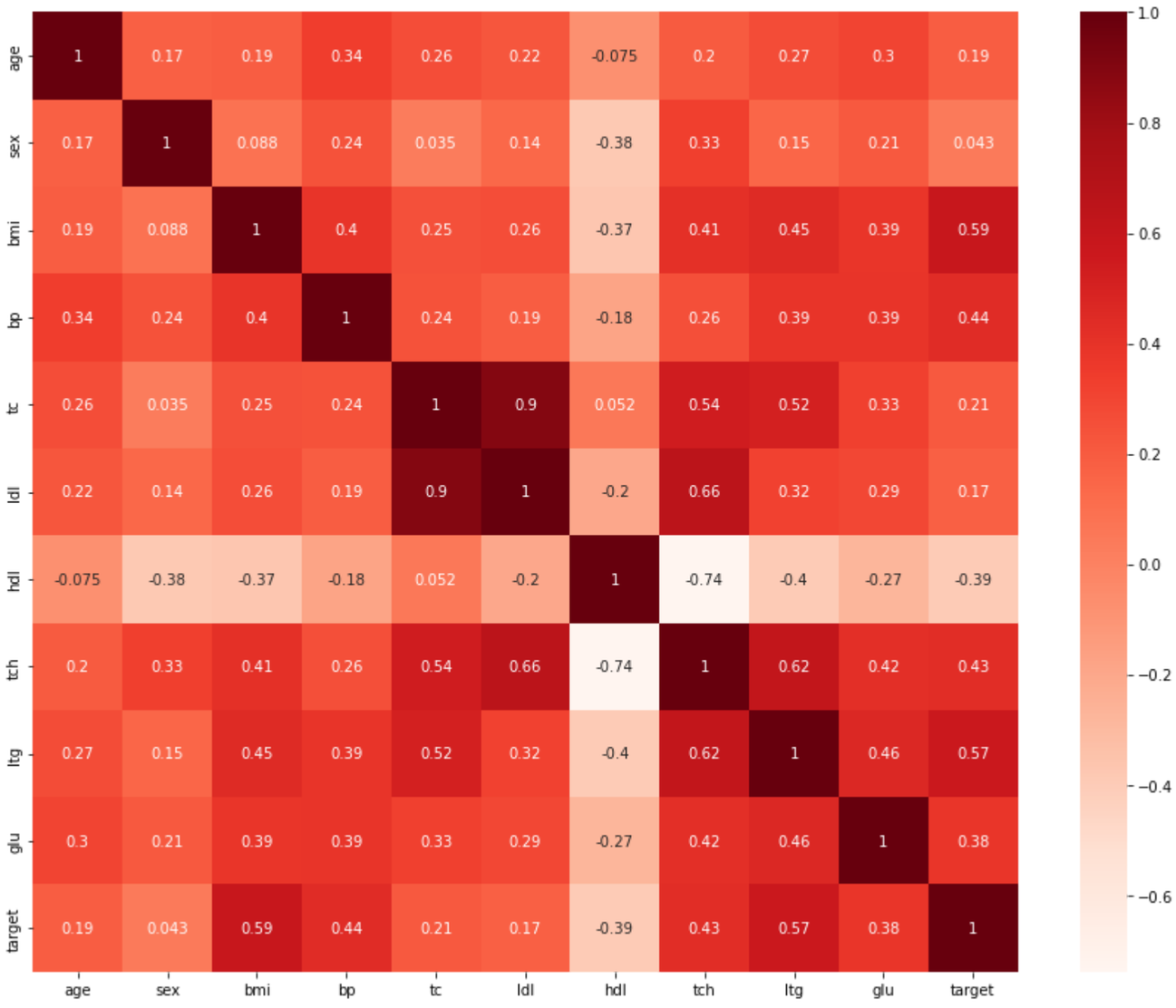
In [592]:
```
fig, ax = plt.subplots(3, 1, figsize=(15, 30))
sns.histplot(x=Diabet.target, data=Diabet, kde=True, element="step",ax=ax[0])
stats.probplot(Diabet.target,plot=ax[1])
sns.boxplot(data=Diabet , x = 'target',ax=ax[2])
```

Out[592]: <AxesSubplot:xlabel='target'>



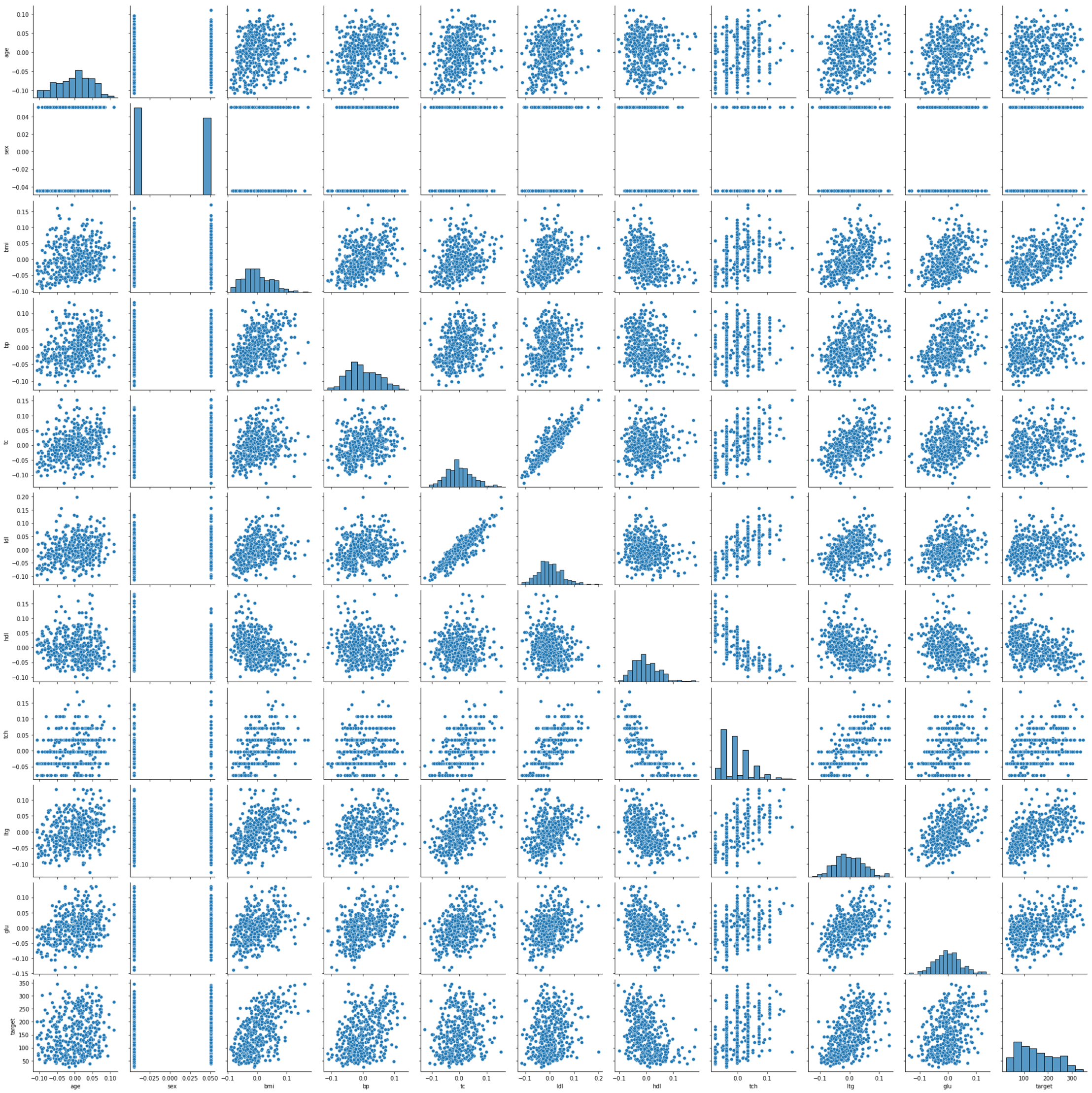
The correlation coefficient has values between -1 to 1
— A value closer to 0 implies weaker correlation (exact 0 implying no correlation)
— A value closer to 1 implies stronger positive correlation
— A value closer to -1 implies stronger negative correlation

In [531]:
```
plt.figure(figsize=(15,12))
cor = Diabet.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



In [532]:
```
1 sns.pairplot(Diabet)
```

Out[532]: <seaborn.axisgrid.PairGrid at 0x202696e7640>



Exercise 5

Implement Linear Regression with Mean Absolute Error as the cost function from scratch. Compare your results with the Linear Regression module of Scikit-Learn.

In [574]:
```
1  class LinearRegressionScrath():
2      #need learning rate for gradient descent and number of iteration
3      def __init__( self, learning_rate, iterations ):
4          self.learning_rate = learning_rate
5          self.iterations = iterations
6
7      #Get training dataset with their labels to fit the model
8      def fit(self,X,y):
9          self.X = X
10         self.y = y
11         #first save the shape of X as m*n shape matrix
12         self.m , self.n = X.shape
13         #then initialize the parameters(we have y = ax + b and our parameters are 'a' and 'b')
14         self.a = np.zeros(self.n)
15         self.b = round(random.uniform(0,1),4)
16         #update weights as many as iteration in a for loop
17         for i in range(self.iterations):
18             self.estimation()
19
20
21
22
23     def estimation(self):
24         Y_pred = np.dot(self.X ,self.a) + self.b
25         dx = sum(np.sign(Y_pred - self.y))/self.n
26         self.a = self.a - self.learning_rate * dx
27         self.b = self.b - self.learning_rate * dx
28
29     def predict(self,X):
30         #predict x just by put in the formula
31          return X.dot( self.a ) + self.b
```

In [595]:
```
Diabet = datasets.load_diabetes(return_X_y=False, as_frame=False, scaled=True)
```

In [597]:
```
X = Diabet.data
Y = Diabet.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
```

In [605]:
```
scratch = LinearRegressionScrath(learning_rate = 0.1, iterations=1500)
scratch.fit( X_train, Y_train)
```

In [606]:
```
 = scratch.predict( X_test )
```

In [610]:
```
mean_absolute_error(Y_pred,Y_test)
```

Out[610]: 51.14095680863134

Linear Regression module of Scikit-Learn

In [611]:
```
 sklearnLinearRegression = LinearRegression()
```

In [612]:
```
sklearnLinearRegression.fit(X_train,Y_train)
```

Out[612]:
```
LinearRegression
LinearRegression()
```

In [613]:
```
sklearnLinearRegression.score(X_train,Y_train)
```

Out[613]: 0.5539250081377072

In [614]:
```
sklearnLinearRegression.score(X_test,Y_test)
```

Out[614]: 0.3322332173106183

train error

In [615]:
```
y_pred_train = sklearnLinearRegression.predict(X_train)
```

In [616]:
```
mean_absolute_error(y_pred_train,Y_train)
```

Out[616]: 42.59334431424345

test error

In [617]:
```
mean_absolute_error(sklearnLinearRegression.predict(X_test),Y_test)
```

Out[617]: 46.17358500370479

Exercise 6

Implement Linear Regression using the normal equation as the training algorithm from scratch.

In [336]:
```
Y = Diabet['target']
X = Diabet.loc[:, Diabet.columns != "target"]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
```

In [334]:
```
def theta(X, y):
    n = X.shape[0]
    X = np.append(X, np.ones((n,1)), axis=1)
    y = y.reshape(n,1)
    theta = np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, y))
    return theta
```

In [335]:
```
# function for predicting our values
def predict(X):

    # add 1 to each row for bias
    X = np.append(X, np.ones((X.shape[0],1)), axis=1)
    prediction = np.dot(X, theta)

    return prediction
```

In [338]:
```
theta = theta(X_train.values,Y_train.values)
```

In [339]:
```
theta
```

Out[339]:
```
array([[ -35.55025079],
       [-243.16508959],
       [ 562.76234744],
       [ 305.46348218],
       [-662.70290089],
       [ 324.20738537],
       [  24.74879489],
       [ 170.3249615 ],
       [ 731.63743545],
       [  43.0309307 ],
       [ 152.53804701]])
```

In [340]:
```
y_pred = predict(X_train)
```

In [342]:
```
mean_absolute_error(y_pred,Y_train)
```

Out[342]: 42.59334431424343

In [347]:
```
mean_squared_error(y_pred,Y_train)
```

Out[347]: 2734.7508990757424

In [343]:
```
y_pred_test = predict(X_test)
```

In [345]:
```
mean_absolute_error(y_pred_test,Y_test)
```

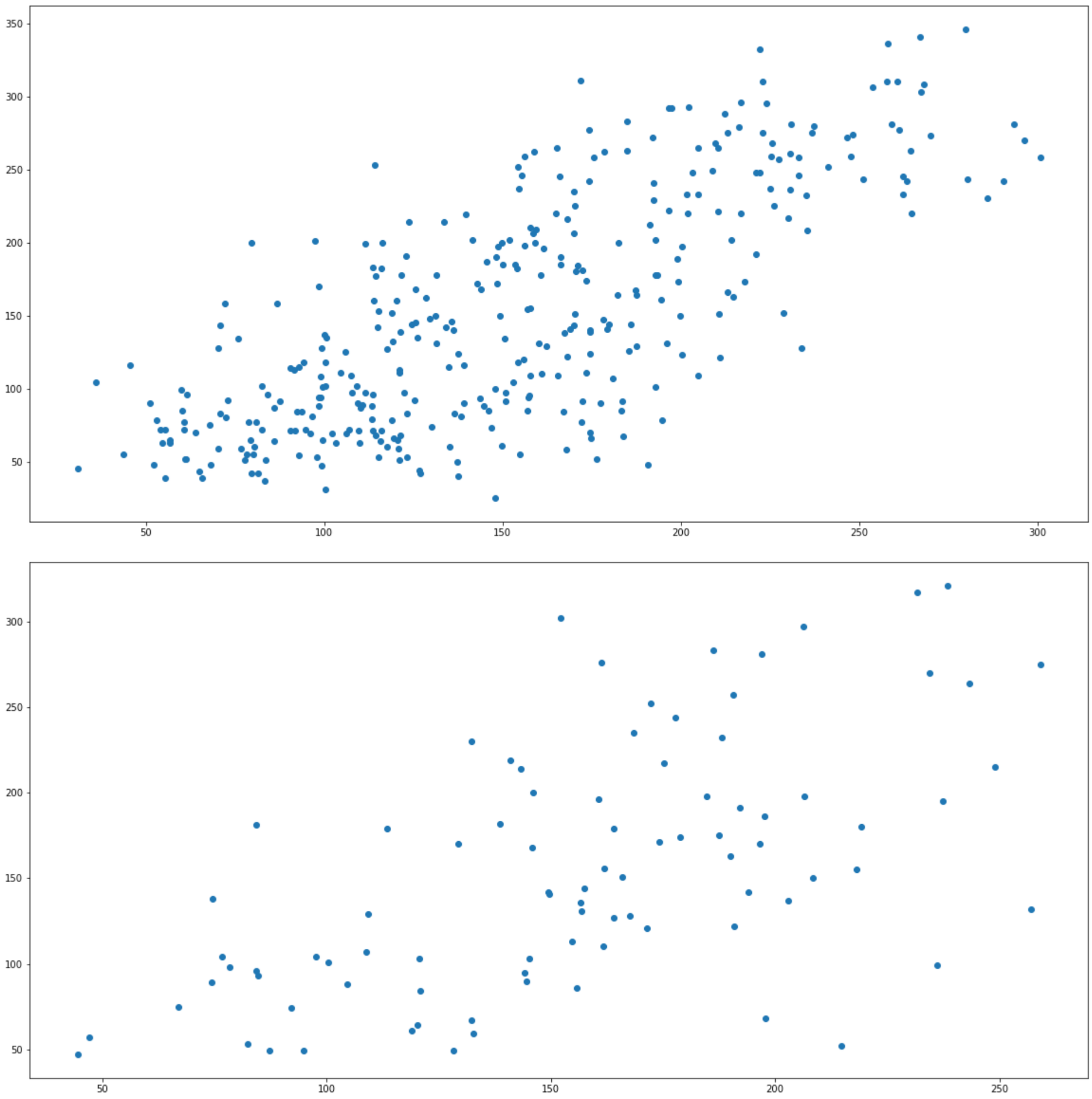
Out[345]: 46.17358500370487

In [346]:
```
mean_squared_error(y_pred_test,Y_test)
```

Out[346]: 3424.2593342986697

In [361]:
```
fig, ax = plt.subplots(1, 1, figsize=(20, 10))
plt.scatter(y_pred ,Y_train)

fig, ax = plt.subplots(1, 1, figsize=(20, 10))
plt.scatter(y_pred_test,Y_test)

```

Out[361]: <matplotlib.collections.PathCollection at 0x291eb9f2b50>



Exercise 10

Implement Forward and Backward Feature selection algorithms from scratch with MSE as the metric.

In [195]:
```
def forward_selection(dataset, n):
    Y = dataset['shares']
    df = dataset.loc[:, dataset.columns != "shares"]
    X_train, X_test, Y_train, Y_test = train_test_split(df, Y, test_size = 0.2, random_state = 0 )
    initial_features = X_train.columns.tolist()
    picked_features = []
    while (len(initial_features) > 0):
        remaining_features = list(set(initial_features) - set(picked_features))

        mse = {}
        for current_feature in remaining_features:
            linear_regression = LinearRegression()
            test_model = X_train[picked_features + [current_feature]].values
            if  test_model.shape[1] == 1:
                test_model = test_model.reshape(-1, 1)
                model = linear_regression.fit(test_model, Y_train.values)
            else:
                model = linear_regression.fit(test_model, Y_train.values)
            y_pred = linear_regression.predict(X_test[picked_features + [current_feature]].values)
            model_error = mean_squared_error(Y_test.values, y_pred)
            mse[current_feature] = model_error

        better_feature = min(mse, key=mse.get)

        if(len(picked_features) < n):
            picked_features.append(better_feature)
        else:
            break

    return picked_features
```

In [196]:
```python
def Forward_SFS(dataset,k):
    Y = dataset['shares']
    df = dataset.loc[:, dataset.columns != "shares"]
    sfs = SFS(LinearRegression(),
          k_features=k,
          forward=True,
          floating=False,
          scoring='neg_mean_squared_error',
          cv = 0)
    sfs.fit(df, Y)
    selceted = sfs.k_feature_names_
    return list(selceted)
```

In [493]:
```python
def backwardSelection(dataset,n):
    Y = dataset['shares']
    df = dataset.loc[:, dataset.columns != "shares"]
    X_train, X_test, Y_train, Y_test = train_test_split(df, Y, test_size = 0.2, random_state = 0 )

    features = X_train.columns.tolist()
    removed = []
    remaining_features = all_features

    for _ in range(len(features)-1):
        remaining_features = list(set(features)-set(removed))
        min_err = math.inf

        for removed_feature in remaining_features:
            our_features = remaining_features.copy()
            our_features.remove(removed_feature)
            X = X_train[our_features].values
            X_t = X_test[our_features].values

            if X.shape[1] == 1:
                X = X.reshape(-1,1)

            model = LinearRegression()
            model.fit(X, Y_train)

            y_pred = model.predict(X_t)
            error = mean_squared_error(Y_test, y_pred)

            if error < min_err:
                min_err = error
                next_removed_feature = removed

        removed.append(next_removed_feature)

    return removed_features[-n:]
```

In [494]:
```python
backwardSelection(dataset, 2)
```

Out[494]: ['num_hrefs', 'LDA_04']

In [497]:
```python
def Backward_SBS(dataset,k):
    Y = dataset['shares']
    df = dataset.loc[:, dataset.columns != "shares"]
    sbs = SFS(LinearRegression(), k_features=k, forward=False, verbose=1, scoring='neg_mean_squared_error')
    sbs.fit(df, Y)
    selceted = sbs.k_feature_names_
    return list(selceted)
```

used in exercise 12 feature selection.wrapper methods :

- Feature Selection

Exercise 12

In this part, you are going to work with the News Popularity Prediction dataset. You will implement a regression model using the Scikit-Learn package to predict the popularity of new articles (the number of times they will be shared online) based on about 60 features. You are expected:

Outline

1. Models

Regression

In [247]:
```python
def simpleLinear(dataset):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    reg = LinearRegression()
    reg.fit(X_train,Y_train)
    Y_pred = reg.predict(X_test)
    score = r2_score(Y_test, Y_pred)
    print("\nLinear Model.....................................\n")
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("RMSE : " + str(math.sqrt(mean_squared_error(Y_test, Y_pred))))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))
```

In [248]:
```
def ridge_regression(dataset,alpha):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    #Fit the model
    ridgereg = Ridge(alpha=alpha)
    ridgereg.fit(X_train,Y_train)
    Y_pred = ridgereg.predict(X_test)
    train_score_ridge = ridgereg.score(X_train, Y_train)
    test_score_ridge = ridgereg.score(X_test, Y_test)
    score = r2_score(Y_test, Y_pred)
    print("\nRidge Model.....................................\n")
    print("The train score for ridge model is {}".format(train_score_ridge))
    print("The test score for ridge model is {}".format(test_score_ridge))
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

In [249]:
```
def lasso_regression(dataset, alpha):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    #Fit the model
    lasso = Lasso(alpha = alpha)
    lasso.fit(X_train,Y_train)
    train_score_ls =lasso.score(X_train,Y_train)
    test_score_ls =lasso.score(X_test,Y_test)
    Y_pred = lasso.predict(X_test)
    score = r2_score(Y_test, Y_pred)
    print("\nLasso Model.....................................\n")
    print("The train score for ls model is {}".format(train_score_ls))
    print("The test score for ls model is {}".format(test_score_ls))
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

In [250]:
```
def polynomial(degree,dataset):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    poly = PolynomialFeatures(degree)
    X_poly = poly.fit_transform(X_train)
    lin2 = LinearRegression()
    lin2.fit(X_poly, Y_train)
    Y_pred = lin2.predict(poly.fit_transform(X_test))
    score = r2_score(Y_test, Y_pred)
    print("\nPolynomial Model.....................................\n")
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("RMSE : " + str(math.sqrt(mean_squared_error(Y_test, Y_pred))))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

In [251]:
```
def randomforestReg(dataset):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    rf = RandomForestRegressor()
    rf.fit(X_train,Y_train)
    Y_pred = rf.predict(X_test)
    score = r2_score(Y_test, Y_pred)
    print("\nRandomForestregressor.....................................\n")
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

In [252]:
```
def gradientBoostingRegressor(dataset):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    gb = GradientBoostingRegressor()
    gb.fit(X_train,Y_train)
    Y_pred = gb.predict(X_test)
    score = r2_score(Y_test, Y_pred)
    print("\nGradientBoostingRegresso.....................................\n")
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

In [253]:
```
def svr(dataset):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    regressor = SVR(kernel = 'rbf')
    regressor.fit(X_train, Y_train)
    Y_pred = regressor.predict(X_test)
    score = r2_score(Y_test, Y_pred)
    print("\nSVR.....................................\n")
    print("MAE : " + str(mean_absolute_error(Y_test, Y_pred)))
    print("MSE : " + str(mean_squared_error(Y_test, Y_pred)))
    print("r2_score : " + str(score) + "  "+ str(round(score, 2) *100))

```

classification

In [254]:
```
def decisionTree(dataset):
    X = dataset[dataset.columns.difference(['target'])]
    Y = dataset['target']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    dtc = DecisionTreeClassifier()
    dtc.fit(X_train, Y_train)
    y_pred = dtc.predict(X_test)
    accuracy = accuracy_score(Y_test, y_pred)
    print("\nDecisionTree.....................................\n")
    print("Accuracy:", accuracy)
    targetNames = dataset['target'].values.unique().tolist()
    print(classification_report(Y_test, y_pred, target_names=targetNames))
```

In [255]:
```
def randomForestClassifier(dataset):
    X = dataset[dataset.columns.difference(['target'])]
    Y = dataset['target']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    rf = RandomForestClassifier(n_estimators=500, random_state=42)
    rf.fit(X_train, Y_train)
    y_pred = rf.predict(X_test)
    accuracy = accuracy_score(Y_test, y_pred)
    print("\nRandomForest.....................................\n")
    print("Accuracy:", accuracy)
    targetNames = dataset['target'].values.unique().tolist()
    print(classification_report(Y_test, y_pred, target_names=targetNames))

```

In [256]:
```
def knn(dataset):
    X = dataset[dataset.columns.difference(['target'])]
    Y = dataset['target']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    knn = KNeighborsClassifier()
    knn.fit(X_train, Y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(Y_test, y_pred)
    print("\nKNN.....................................\n")
    print("Accuracy:", accuracy)
    targetNames = dataset['target'].values.unique().tolist()
    print(classification_report(Y_test, y_pred, target_names=targetNames))
```

In [257]:
```
def svc(dataset):
    X = dataset[dataset.columns.difference(['target'])]
    Y = dataset['target']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    clf = SVC(kernel='linear')
    clf.fit(X_train,Y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(Y_test, y_pred)
    print("\nSVC.....................................\n")
    print("Accuracy:", accuracy)
    targetNames = dataset['target'].values.unique().tolist()
    print(classification_report(Y_test, y_pred, target_names=targetNames))
```

2. Perform exploratory data analysis on the dataset.

In [638]:
```
OnlineNewsPopularity = pd.read_csv("dataset/OnlineNewsPopularity.csv")
```

In [639]:
```
OnlineNewsPopularity.shape
```

Out[639]: (39644, 61)

In [640]:
```
OnlineNewsPopularity.head(10)
```

Out[640]:

	url	timedelta	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_l
0	http://mashable.com/2013/01/07/amazon-instant-video-browser/	731.0	12.0	219.0	0.663594	1.0	0.815385	4.0	2.0	1.0	0.0	4.6
1	http://mashable.com/2013/01/07/ap-samsung-sponsored-tweets/	731.0	9.0	255.0	0.604743	1.0	0.791946	3.0	1.0	1.0	0.0	4.9
2	http://mashable.com/2013/01/07/apple-40-billion-app-downloads/	731.0	9.0	211.0	0.575130	1.0	0.663866	3.0	1.0	1.0	0.0	4.3
3	http://mashable.com/2013/01/07/astronaut-notre-dame-bcs/	731.0	9.0	531.0	0.503788	1.0	0.665635	9.0	0.0	1.0	0.0	4.4
4	http://mashable.com/2013/01/07/att-u-verse-apps/	731.0	13.0	1072.0	0.415646	1.0	0.540890	19.0	19.0	20.0	0.0	4.6
5	http://mashable.com/2013/01/07/beewi-smart-toys/	731.0	10.0	370.0	0.559889	1.0	0.698198	2.0	2.0	0.0	0.0	4.3
6	http://mashable.com/2013/01/07/bodymedia-armbandgets-update/	731.0	8.0	960.0	0.418163	1.0	0.549834	21.0	20.0	20.0	0.0	4.6
7	http://mashable.com/2013/01/07/canon-poweshot-n/	731.0	12.0	989.0	0.433574	1.0	0.572108	20.0	20.0	20.0	0.0	4.6
8	http://mashable.com/2013/01/07/car-of-the-future-infographic/	731.0	11.0	97.0	0.670103	1.0	0.836735	2.0	0.0	0.0	0.0	4.8
9	http://mashable.com/2013/01/07/chuck-hagel-website/	731.0	10.0	231.0	0.636364	1.0	0.797101	4.0	1.0	1.0	1.0	5.0

In [261]:
```
#Removing Space Character from Feature names
OnlineNewsPopularity.columns=OnlineNewsPopularity.columns.str.replace(" ","")
```

It's obvious that we don't need URL and timedelta(non predictive)

In [262]:
```
OnlineNewsPopularity.drop('url', inplace=True, axis=1)
OnlineNewsPopularity.drop('timedelta', inplace=True, axis=1)
```

Checking null values and data types

In [263]:
```
1 OnlineNewsPopularity.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39644 entries, 0 to 39643
Data columns (total 59 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   n_tokens_title                 39644 non-null  float64
 1   n_tokens_content               39644 non-null  float64
 2   n_unique_tokens                39644 non-null  float64
 3   n_non_stop_words               39644 non-null  float64
 4   n_non_stop_unique_tokens       39644 non-null  float64
 5   num_hrefs                      39644 non-null  float64
 6   num_self_hrefs                 39644 non-null  float64
 7   num_imgs                       39644 non-null  float64
 8   num_videos                     39644 non-null  float64
 9   average_token_length           39644 non-null  float64
 10  num_keywords                   39644 non-null  float64
 11  data_channel_is_lifestyle      39644 non-null  float64
 12  data_channel_is_entertainment  39644 non-null  float64
 13  data_channel_is_bus            39644 non-null  float64
 14  data_channel_is_socmed         39644 non-null  float64
 15  data_channel_is_tech           39644 non-null  float64
 16  data_channel_is_world          39644 non-null  float64
 17  kw_min_min                     39644 non-null  float64
 18  kw_max_min                     39644 non-null  float64
 19  kw_avg_min                     39644 non-null  float64
 20  kw_min_max                     39644 non-null  float64
 21  kw_max_max                     39644 non-null  float64
 22  kw_avg_max                     39644 non-null  float64
 23  kw_min_avg                     39644 non-null  float64
 24  kw_max_avg                     39644 non-null  float64
 25  kw_avg_avg                     39644 non-null  float64
 26  self_reference_min_shares      39644 non-null  float64
 27  self_reference_max_shares      39644 non-null  float64
 28  self_reference_avg_sharess     39644 non-null  float64
 29  weekday_is_monday              39644 non-null  float64
 30  weekday_is_tuesday             39644 non-null  float64
 31  weekday_is_wednesday           39644 non-null  float64
 32  weekday_is_thursday            39644 non-null  float64
 33  weekday_is_friday              39644 non-null  float64
 34  weekday_is_saturday            39644 non-null  float64
 35  weekday_is_sunday              39644 non-null  float64
 36  is_weekend                     39644 non-null  float64
 37  LDA_00                         39644 non-null  float64
 38  LDA_01                         39644 non-null  float64
 39  LDA_02                         39644 non-null  float64
 40  LDA_03                         39644 non-null  float64
 41  LDA_04                         39644 non-null  float64
 42  global_subjectivity            39644 non-null  float64
 43  global_sentiment_polarity      39644 non-null  float64
 44  global_rate_positive_words     39644 non-null  float64
 45  global_rate_negative_words     39644 non-null  float64
 46  rate_positive_words            39644 non-null  float64
 47  rate_negative_words            39644 non-null  float64
 48  avg_positive_polarity          39644 non-null  float64
 49  min_positive_polarity          39644 non-null  float64
 50  max_positive_polarity          39644 non-null  float64
 51  avg_negative_polarity          39644 non-null  float64
 52  min_negative_polarity          39644 non-null  float64
 53  max_negative_polarity          39644 non-null  float64
 54  title_subjectivity             39644 non-null  float64
 55  title_sentiment_polarity       39644 non-null  float64
 56  abs_title_subjectivity         39644 non-null  float64
 57  abs_title_sentiment_polarity   39644 non-null  float64
 58  shares                         39644 non-null  int64  
dtypes: float64(58), int64(1)
memory usage: 17.8 MB
```

Checking Duplicate values

In [264]:
```
1 OnlineNewsPopularity=OnlineNewsPopularity.drop_duplicates()
2 OnlineNewsPopularity.shape
```

Out[264]: (39644, 59)

In [265]:
```
1 X = OnlineNewsPopularity[OnlineNewsPopularity.columns.difference(['shares'])]
2 Y = OnlineNewsPopularity['shares']
```

In [266]:
```
1 OnlineNewsPopularity.describe()
```

Out[266]:

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	data_channel_is_l
count	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.
mean	10.398749	546.514731	0.548216	0.996469	0.689175	10.883690	3.293638	4.544143	1.249874	4.548239	7.223767	0.
std	2.114037	471.107508	3.520708	5.231231	3.264816	11.332017	3.855141	8.309434	4.107855	0.844406	1.909130	0.
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.
25%	9.000000	246.000000	0.470870	1.000000	0.625739	4.000000	1.000000	1.000000	0.000000	4.478404	6.000000	0.
50%	10.000000	409.000000	0.539226	1.000000	0.690476	8.000000	3.000000	1.000000	0.000000	4.664082	7.000000	0.
75%	12.000000	716.000000	0.608696	1.000000	0.754630	14.000000	4.000000	4.000000	1.000000	4.854839	9.000000	0.
max	23.000000	8474.000000	701.000000	1042.000000	650.000000	304.000000	116.000000	128.000000	91.000000	8.041534	10.000000	1.

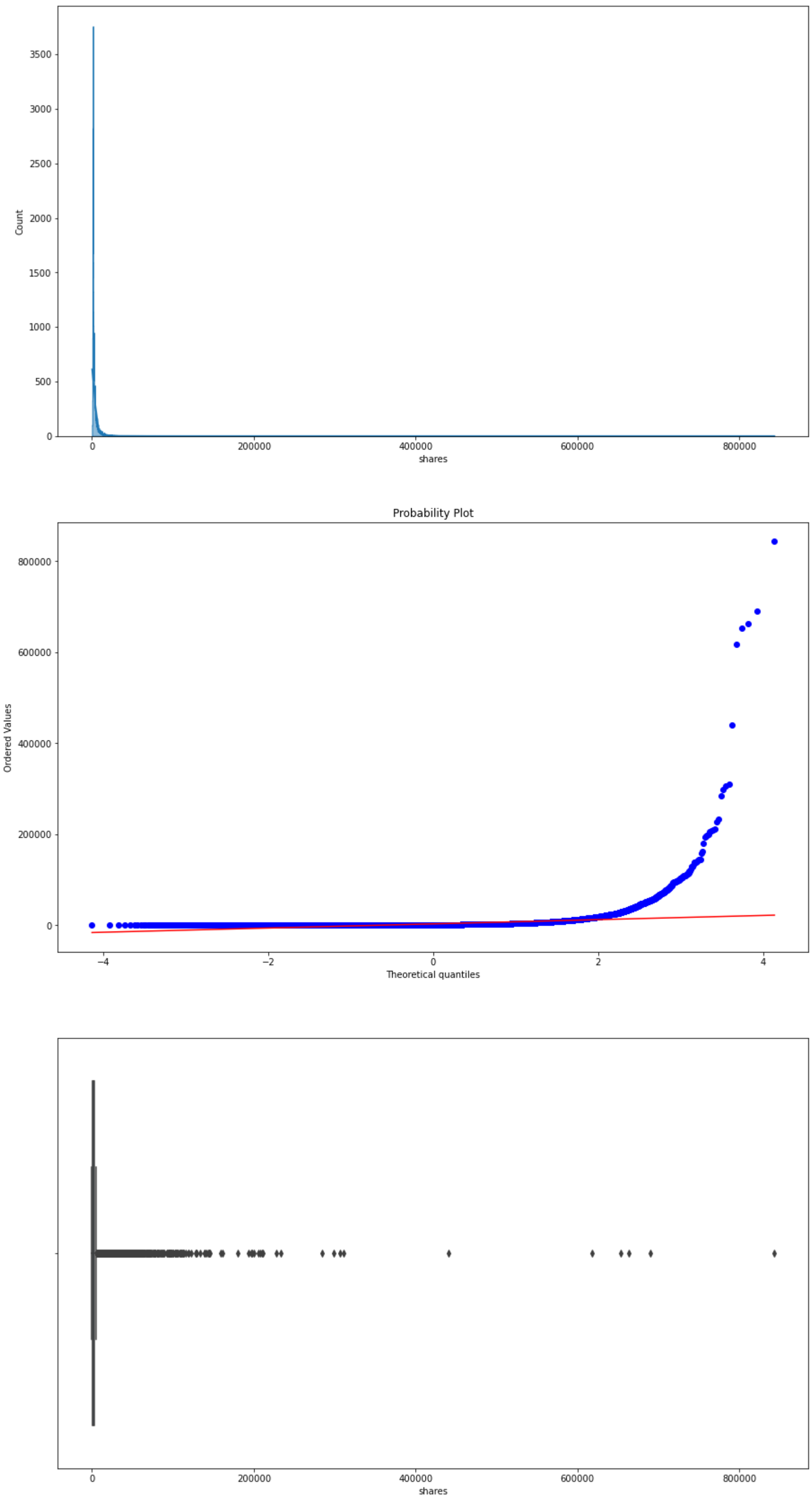
In [267]:
```
1 Y.describe()
```

Out[267]:
```
count     39644.000000
mean       3395.380184
std       11626.950749
min           1.000000
25%         946.000000
50%        1400.000000
75%        2800.000000
max      843300.000000
Name: shares, dtype: float64
```

high std

In [268]:
```
fig, ax = plt.subplots(3, 1, figsize=(15, 30))
sns.histplot(x=OnlineNewsPopularity.loc[:,'shares'], data=OnlineNewsPopularity, kde=True, element="step",ax=ax[0])
stats.probplot(OnlineNewsPopularity.loc[:,'shares'],plot=ax[1])
sns.boxplot(data=OnlineNewsPopularity , x = 'shares' ,ax=ax[2])
```

Out[268]: <AxesSubplot:xlabel='shares'>



As we expected . . .

In [269]:
```
#testin the first datasets with our models
```

In [270]:
```
simpleLinear(OnlineNewsPopularity)
ridge_regression(OnlineNewsPopularity,10)
lasso_regression(OnlineNewsPopularity,10)
```

```
Linear Model.....................................

MAE : 60745495.04133717
MSE : 2.925520993211126e+19
RMSE : 5408808550.144039
r2_score : -384409419567.89185  -38440941956789.0

Ridge Model.....................................

The train score for ridge model is 0.021939678798331474
The test score for ridge model is -0.3358153134312516
MAE : 3085.732251457612
MSE : 101661289.85285343
r2_score : -0.3358153134312516  -34.0

Lasso Model.....................................

The train score for ls model is 0.02129886307728668
The test score for ls model is 0.024839810969102483
MAE : 3022.1138232510402
MSE : 74213883.93533753
r2_score : 0.024839810969102483  2.0
```

In [271]:
```
polynomial(2,OnlineNewsPopularity)
```

```
Polynomial Model.................................

MAE : 933833.1928336386
MSE : 6853294077431953.0
RMSE : 82784624.64390326
r2_score : -90051337.01992922  -9005133702.0
```

In this case technique of using log(target) instead of target is very effective in reducing the workload on our models.

In [272]:
```
#OnlineNewsPopularity saved as orginal data
dataset = OnlineNewsPopularity.copy()
```

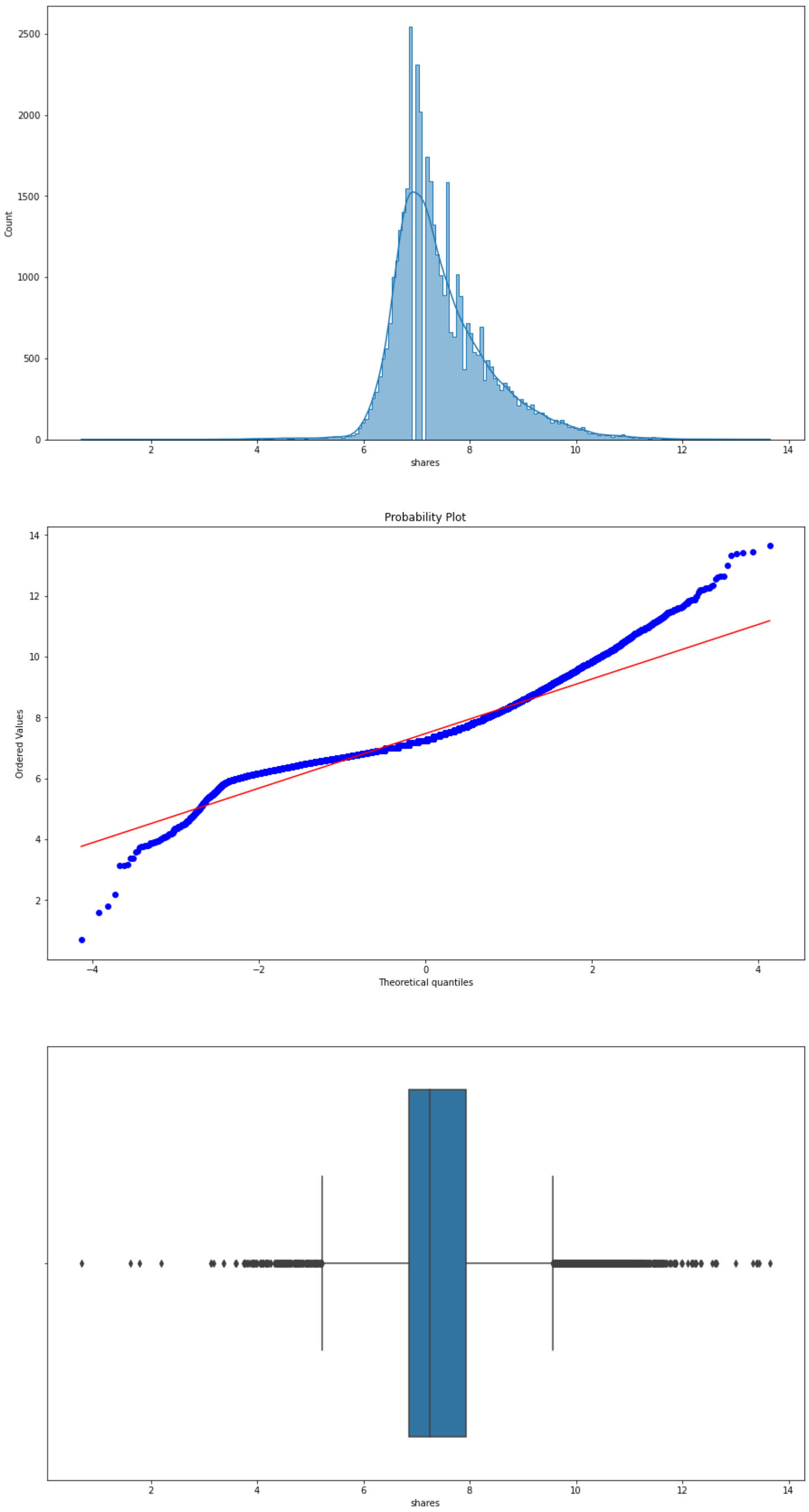
In [273]:
```
dataset['shares'] = np.log1p(OnlineNewsPopularity.shares)
```

In [274]:
```
dataset['shares'].describe()
```

Out[274]:
```
count    39644.000000
mean         7.475692
std          0.929674
min          0.693147
25%          6.853299
50%          7.244942
75%          7.937732
max         13.645079
Name: shares, dtype: float64
```

In [275]:
```
fig, ax = plt.subplots(3, 1, figsize=(15, 30))
sns.histplot(x=dataset.loc[:,'shares'], data=dataset, kde=True, element="step",ax=ax[0])
stats.probplot(dataset.loc[:,'shares'],plot=ax[1])
sns.boxplot(data=dataset , x = 'shares' ,ax=ax[2])
```

Out[275]: <AxesSubplot:xlabel='shares'>



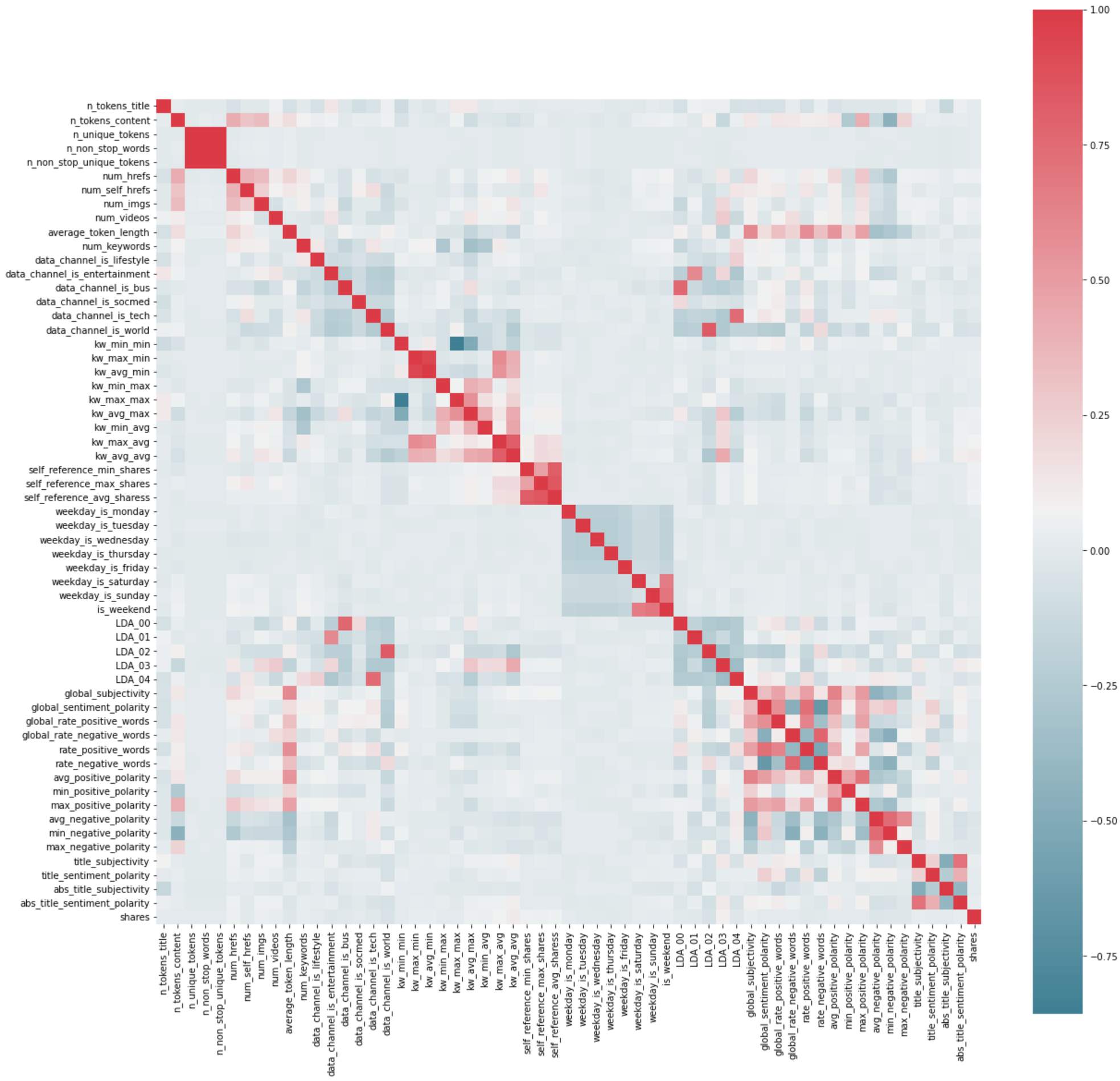
STD decrease well from 11626.950749 to 0.929674 so we observe normal distribution and this helps our future model(s) to predict better

In [276]:
```
# Plot correlation matrix for first dataset
```

In [277]:
```
f, ax = plt.subplots(figsize=(20, 20))
corr1 = OnlineNewsPopularity.corr()
sns.heatmap(corr1, mask=np.zeros_like(corr1, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
```

Out[277]: <AxesSubplot:>



In [278]:
```
1 print(abs(corr1["shares"]).sort_values(ascending=False))
```

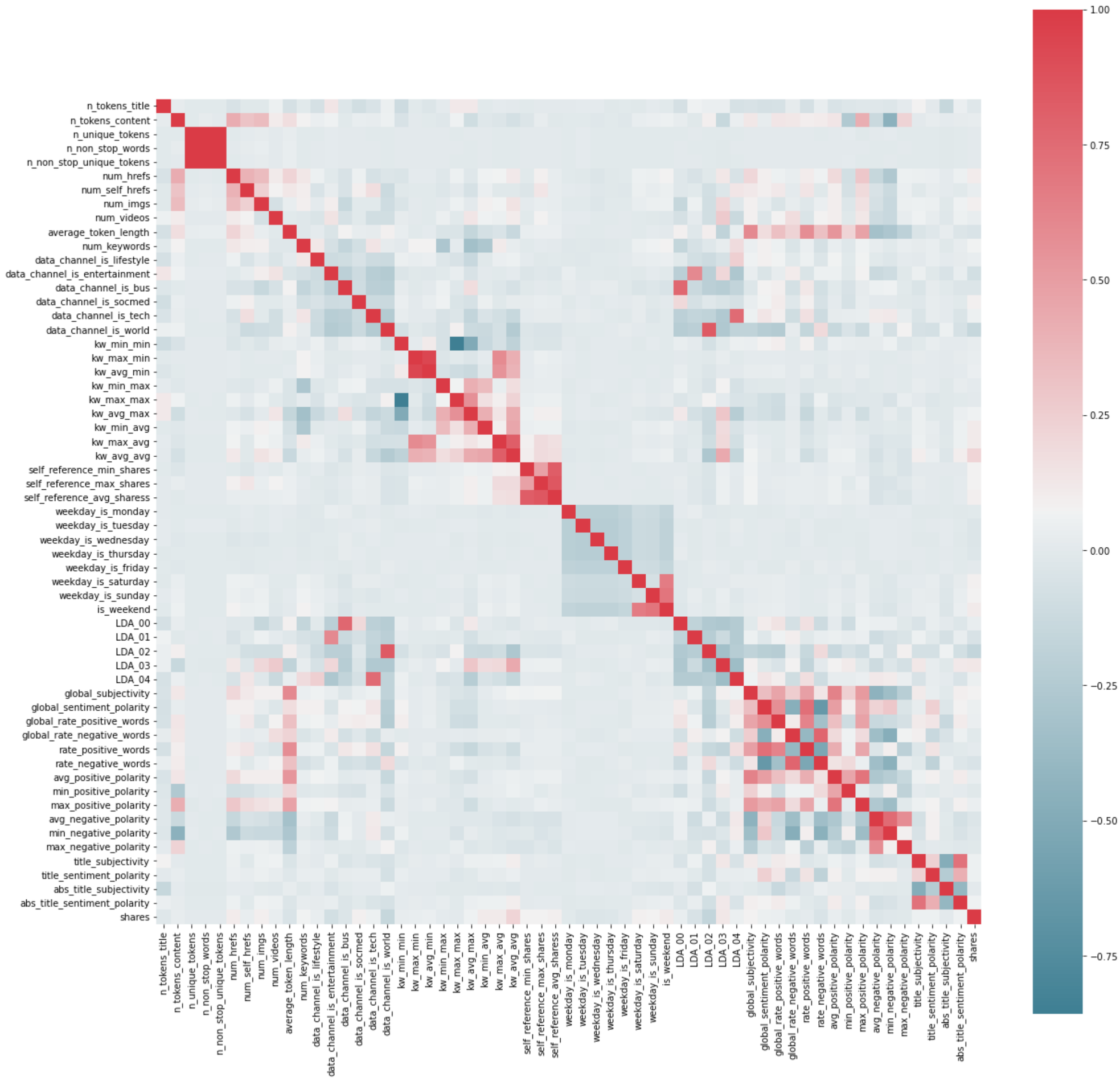
```
shares                          1.000000
kw_avg_avg                      0.110413
LDA_03                          0.083771
kw_max_avg                      0.064306
LDA_02                          0.059163
self_reference_avg_sharess      0.057789
self_reference_min_shares       0.055958
data_channel_is_world           0.049497
self_reference_max_shares       0.047115
num_hrefs                       0.045404
kw_avg_max                      0.044686
kw_min_avg                      0.039551
num_imgs                        0.039388
avg_negative_polarity           0.032029
global_subjectivity             0.031604
kw_avg_min                      0.030406
kw_max_min                      0.030114
abs_title_sentiment_polarity    0.027135
num_videos                      0.023936
average_token_length            0.022007
title_subjectivity              0.021967
num_keywords                    0.021818
max_negative_polarity           0.019300
min_negative_polarity           0.019297
data_channel_is_entertainment   0.017006
is_weekend                      0.016958
LDA_04                          0.016622
weekday_is_saturday             0.015082
data_channel_is_tech            0.013253
rate_positive_words             0.013241
title_sentiment_polarity        0.012772
data_channel_is_bus             0.012376
avg_positive_polarity           0.012142
LDA_01                          0.010183
max_positive_polarity           0.010068
weekday_is_monday               0.009726
weekday_is_thursday             0.008833
n_tokens_title                  0.008783
weekday_is_sunday               0.008230
weekday_is_tuesday              0.007941
kw_max_max                      0.007863
global_rate_negative_words      0.006615
data_channel_is_lifestyle       0.005831
rate_negative_words             0.005183
data_channel_is_socmed          0.005021
global_sentiment_polarity       0.004163
kw_min_max                      0.003901
weekday_is_friday               0.003884
weekday_is_wednesday            0.003801
LDA_00                          0.003793
n_tokens_content                0.002459
num_self_hrefs                  0.001900
abs_title_subjectivity          0.001481
kw_min_min                      0.001051
n_unique_tokens                 0.000806
global_rate_positive_words      0.000543
n_non_stop_words                0.000443
n_non_stop_unique_tokens        0.000114
min_positive_polarity           0.000040
Name: shares, dtype: float64
```

In [279]:
```
1 # Now plot heatmap of correlation for "dataset"(with log shares as target)
```

In [280]:
```
f, ax = plt.subplots(figsize=(20, 20))
corr2 = dataset.corr()
sns.heatmap(corr2, mask=np.zeros_like(corr2, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
```

Out[280]: <AxesSubplot:>



In [281]:
```
print(abs(corr2["shares"]).sort_values(ascending=False))
```

```
shares                          1.000000
kw_avg_avg                      0.221822
LDA_02                          0.165076
data_channel_is_world           0.151685
LDA_03                          0.126037
is_weekend                      0.114391
kw_max_avg                      0.109343
kw_min_avg                      0.108850
num_hrefs                       0.105142
num_imgs                        0.091519
self_reference_avg_sharess      0.087238
data_channel_is_entertainment   0.082531
data_channel_is_socmed          0.081144
weekday_is_saturday             0.079158
self_reference_max_shares       0.077442
weekday_is_sunday               0.076969
global_subjectivity             0.075174
self_reference_min_shares       0.073374
num_keywords                    0.065945
kw_avg_max                      0.059678
abs_title_sentiment_polarity    0.059191
data_channel_is_tech            0.054539
LDA_01                          0.052992
title_subjectivity              0.052712
title_sentiment_polarity        0.051038
global_sentiment_polarity       0.048927
rate_negative_words             0.048176
average_token_length            0.045321
LDA_04                          0.044744
global_rate_positive_words      0.040969
kw_avg_min                      0.039539
weekday_is_wednesday            0.036609
max_positive_polarity           0.035160
avg_positive_polarity           0.034460
avg_negative_polarity           0.034323
data_channel_is_lifestyle       0.033228
kw_max_min                      0.033104
num_videos                      0.032151
weekday_is_tuesday              0.031095
data_channel_is_bus             0.030587
LDA_00                          0.030577
num_self_hrefs                  0.029113
weekday_is_thursday             0.028169
n_tokens_content                0.024846
min_negative_polarity           0.022748
kw_min_min                      0.022556
n_tokens_title                  0.019048
max_negative_polarity           0.014204
min_positive_polarity           0.010771
kw_min_max                      0.010531
rate_positive_words             0.008996
weekday_is_monday               0.008238
global_rate_negative_words      0.006029
n_non_stop_words                0.005407
weekday_is_friday               0.005113
n_unique_tokens                 0.004972
n_non_stop_unique_tokens        0.003710
kw_max_max                      0.002099
abs_title_subjectivity          0.000762
Name: shares, dtype: float64
```

In [282]:
```
dataset.describe()
```

Out[282]:

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	data_channel_is_l
count	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.
mean	10.398749	546.514731	0.548216	0.996469	0.689175	10.883690	3.293638	4.544143	1.249874	4.548239	7.223767	0.
std	2.114037	471.107508	3.520708	5.231231	3.264816	11.332017	3.855141	8.309434	4.107855	0.844406	1.909130	0.
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.
25%	9.000000	246.000000	0.470870	1.000000	0.625739	4.000000	1.000000	1.000000	0.000000	4.478404	6.000000	0.
50%	10.000000	409.000000	0.539226	1.000000	0.690476	8.000000	3.000000	1.000000	0.000000	4.664082	7.000000	0.
75%	12.000000	716.000000	0.608696	1.000000	0.754630	14.000000	4.000000	4.000000	1.000000	4.854839	9.000000	0.
max	23.000000	8474.000000	701.000000	1042.000000	650.000000	304.000000	116.000000	128.000000	91.000000	8.041534	10.000000	1.

In [283]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 241.14547330187565
MSE : 458621856.91970253
RMSE : 21415.458363521022
r2_score : -534333453.04510975  -53433345305.0

Ridge Model.....................................

The train score for ridge model is 0.12816800456595312
The test score for ridge model is -0.07899986223920963
MAE : 0.6482271165314156
MSE : 0.9261125551657352
r2_score : -0.07899986223920963  -8.0

Lasso Model.....................................

The train score for ls model is 0.08370922882846588
The test score for ls model is 0.07482549346511824
MAE : 0.6687128272738687
MSE : 0.7940832582157135
r2_score : 0.07482549346511824  7.000000000000001
```

In [284]:
```
polynomial(2,dataset)
```

```
Polynomial Model.....................................

MAE : 617.5197701083538
MSE : 3017197304.5708523
RMSE : 54929.020604511534
r2_score : -3515291372.409636  -351529137241.0
```

Based on this if both of these features are equals to zero it's better to delete them

In [285]:
```
#save the previous dataaset
dataset_withContetn = dataset.copy()
```

2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content

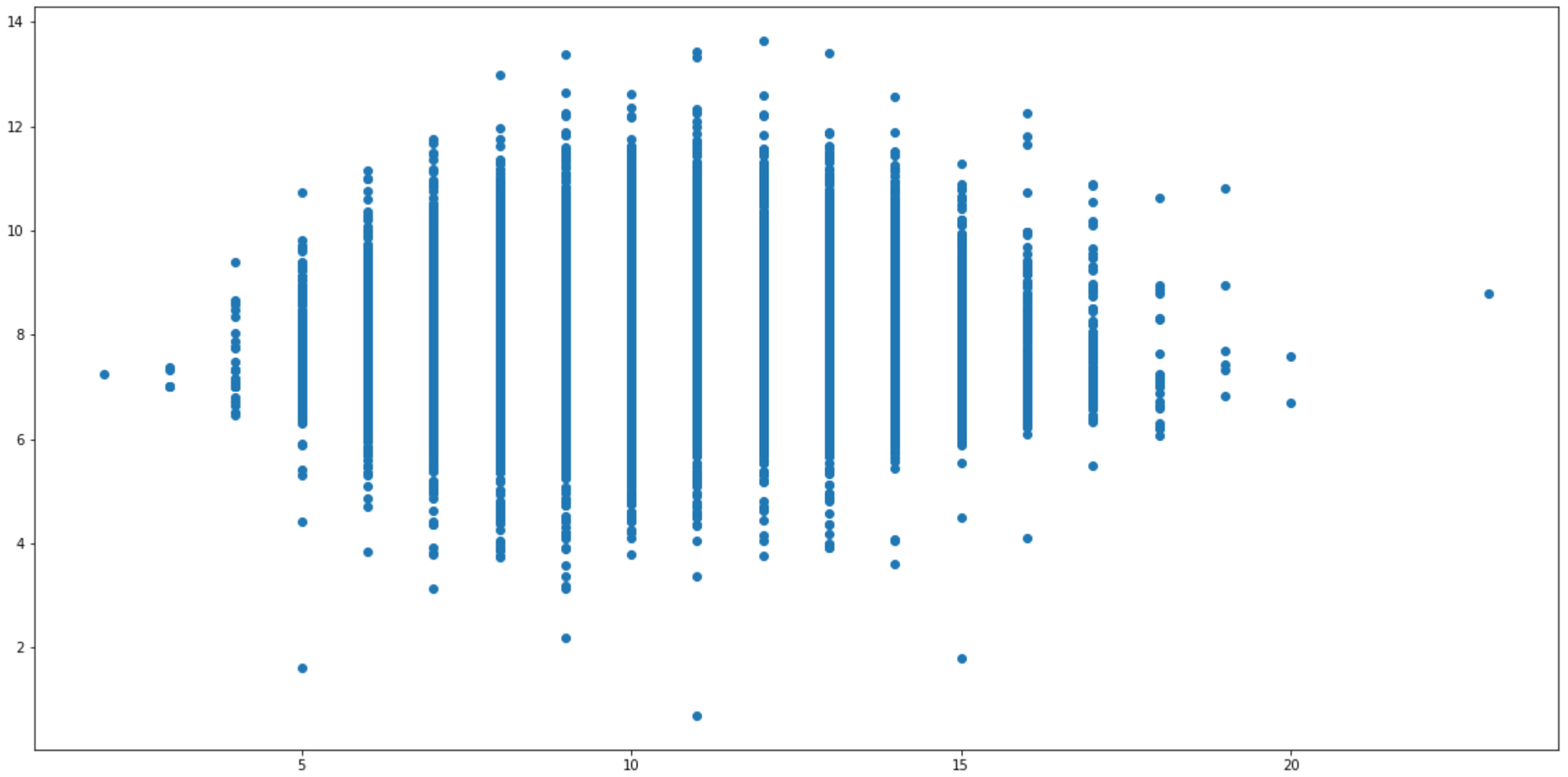
In [286]:
```
num=dataset[dataset['n_tokens_content']==0].index
title =dataset[dataset['n_tokens_title']==0].index
print('number of news items with no words',num.size)
print('number of news items with no words in title',title.size)
```

```
number of news items with no words 1181
number of news items with no words in title 0
```

well all of them have title but 1181 of them doesn't have any content well I get a sample of data and dropping to test the result

In [287]:
```
plt.figure(figsize=(20,10))
plt.scatter(dataset['n_tokens_title'] ,dataset['shares'] )
```

Out[287]: <matplotlib.collections.PathCollection at 0x20202b4ea60>



In [288]:
```
df = dataset[dataset['n_tokens_content'] == 0]
```

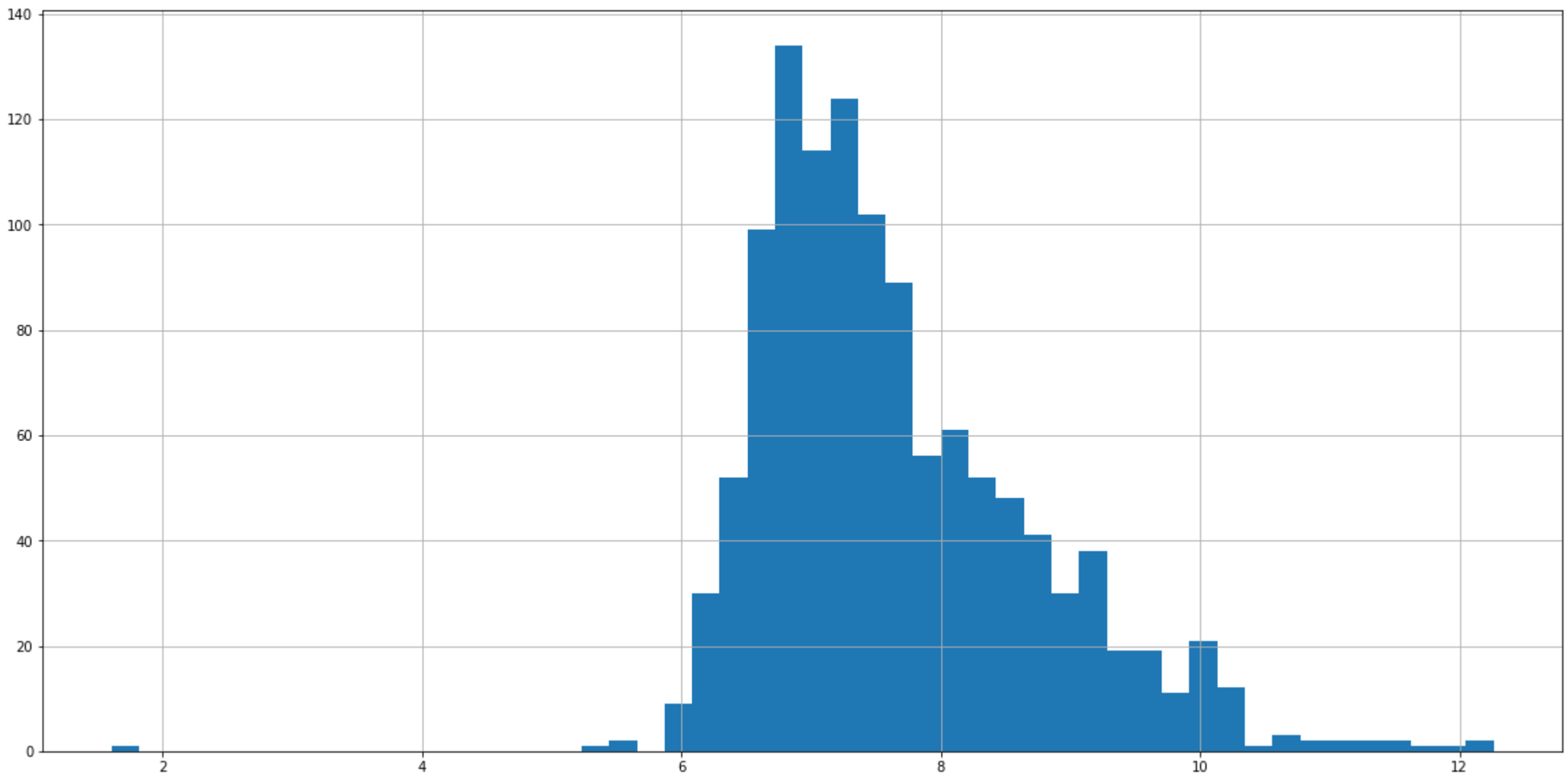
In [289]:
```
def correlation(feature):
    print(str(feature)  + " Spearman's  : " +str(stats.spearmanr(dataset[feature] , dataset['shares']).correlation))
    print(str(feature)  + " kendall  : " + str(stats.kendalltau(dataset[feature] , dataset['shares']).correlation))
```

In [290]:
```
correlation('n_tokens_title')
```

```
n_tokens_title Spearman's  : -0.040278201936068875
n_tokens_title kendall  : -0.028824350176953673
```

In [291]:
```
plt.figure(figsize=(20,10))
df['shares'].hist(bins=50)
```

Out[291]: <AxesSubplot:>



In [292]:
```
# drop data with zero content
```

In [293]:
```
dataset = dataset[dataset['n_tokens_content'] != 0]
```

In [294]:
```
dataset.shape
```

Out[294]: (38463, 59)

In [295]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.6361076401522979
MSE : 0.7365150240927806
RMSE : 0.858204535115482
r2_score : 0.10971945662174454  11.0

Ridge Model.....................................

The train score for ridge model is 0.133441885859363
The test score for ridge model is 0.10977360536822334
MAE : 0.6359250264231455
MSE : 0.7364702276905519
r2_score : 0.10977360536822334  11.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001
```

In [296]:
```
polynomial(2,dataset)
```

```
Polynomial Model.................................

MAE : 0.6462615040902566
MSE : 0.880723515157697
RMSE : 0.9384687076070768
r2_score : -0.06459608289243524  -6.0
```

Categorical Feature Encoding

In [297]:
```
#save the previous dataset
dataset_Yhat = dataset.copy()
```

In [298]:
```
publishdayMerge=dataset[['weekday_is_monday','weekday_is_tuesday','weekday_is_wednesday',
                         'weekday_is_thursday', 'weekday_is_friday','weekday_is_saturday' ,'weekday_is_sunday']]
temp_arr=[]
for r in list(range(publishdayMerge.shape[0])):
    for c in list(range(publishdayMerge.shape[1])):
        if ((c==0) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(1)
        elif ((c==1) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(2)
        elif ((c==2) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(3)
        elif ((c==3) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(4)
        elif ((c==4) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(5)
        elif ((c==5) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(6)
        elif ((c==6) and (publishdayMerge.iloc[r,c])==1):
            temp_arr.append(7)
```

In [299]:
```
dataset.insert(loc=11, column='weekdays', value=temp_arr)
```

In [300]:
```
dataset.drop(labels=['weekday_is_monday','weekday_is_tuesday','weekday_is_wednesday',
                     'weekday_is_thursday', 'weekday_is_friday','weekday_is_saturday' ,'weekday_is_sunday'], axis = 1, inplace=True)
```

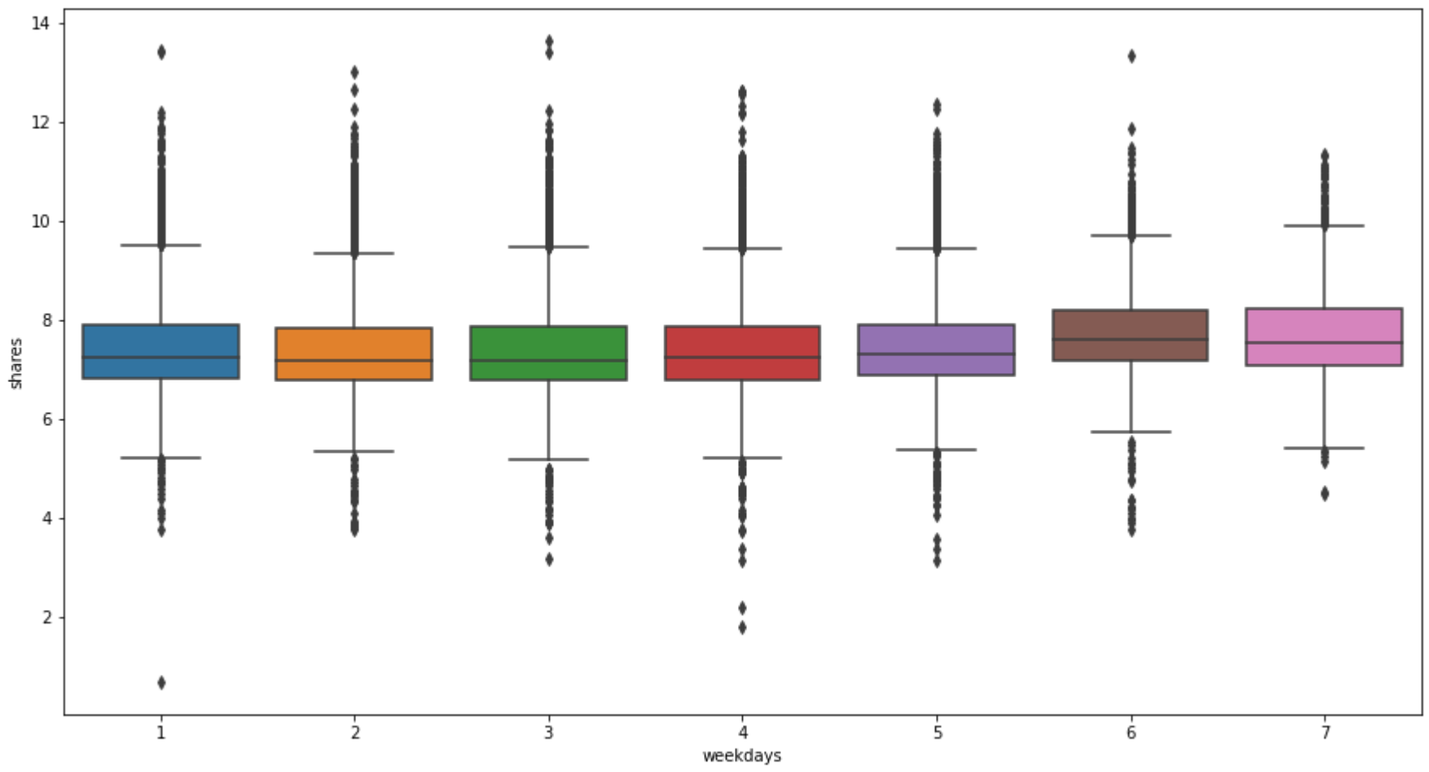
In [301]:
```
DataChannelMerge=dataset[['data_channel_is_lifestyle','data_channel_is_entertainment' ,'data_channel_is_bus',
                          'data_channel_is_socmed' ,'data_channel_is_tech','data_channel_is_world' ]]
#Logic to merge data channel
DataChannel_arr=[]
for r in list(range(DataChannelMerge.shape[0])):
    if (((DataChannelMerge.iloc[r,0])==0) and ((DataChannelMerge.iloc[r,1])==0) and ((DataChannelMerge.iloc[r,2])==0) and ((DataChannelMerge.iloc[r,3])==0) and ((DataChannelM
        DataChannel_arr.append(0)
    for c in list(range(DataChannelMerge.shape[1])):
        if ((c==0) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(1)
        elif ((c==1) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(2)
        elif ((c==2) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(3)
        elif ((c==3) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(4)
        elif ((c==4) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(5)
        elif ((c==5) and (DataChannelMerge.iloc[r,c])==1):
            DataChannel_arr.append(6)
```

In [302]:
```
dataset.insert(loc=12, column='data_channel', value=DataChannel_arr)
```

In [303]:
```
dataset.drop(labels=['data_channel_is_lifestyle','data_channel_is_entertainment' ,'data_channel_is_bus',
                     'data_channel_is_socmed' ,'data_channel_is_tech','data_channel_is_world' ], axis = 1, inplace=True)
```

In [304]:
```
1 plt.figure(figsize=(15,8))
2 sns.boxplot(data=dataset , x = 'weekdays' , y = 'shares')
```

Out[304]: <AxesSubplot:xlabel='weekdays', ylabel='shares'>



In [305]:
```
1 dataset
```

Out[305]:

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channe
0	12.0	219.0	0.663594	1.0	0.815385	4.0	2.0	1.0	0.0	4.680365	5.0	1	2
1	9.0	255.0	0.604743	1.0	0.791946	3.0	1.0	1.0	0.0	4.913725	4.0	1	3
2	9.0	211.0	0.575130	1.0	0.663866	3.0	1.0	1.0	0.0	4.393365	6.0	1	3
3	9.0	531.0	0.503788	1.0	0.665635	9.0	0.0	1.0	0.0	4.404896	7.0	1	2
4	13.0	1072.0	0.415646	1.0	0.540890	19.0	19.0	20.0	0.0	4.682836	7.0	1	5
...	...	...	...	...	...	...	...	...	...	...	...	...	..
39639	11.0	346.0	0.529052	1.0	0.684783	9.0	7.0	1.0	1.0	4.523121	8.0	3	5
39640	12.0	328.0	0.696296	1.0	0.885057	9.0	7.0	3.0	48.0	4.405488	7.0	3	4
39641	10.0	442.0	0.516355	1.0	0.644128	24.0	1.0	12.0	1.0	5.076923	8.0	3	0
39642	6.0	682.0	0.539493	1.0	0.692661	10.0	1.0	1.0	0.0	4.975073	5.0	3	6
39643	10.0	157.0	0.701987	1.0	0.846154	1.0	1.0	0.0	2.0	4.471338	4.0	3	2

38463 rows × 48 columns

convert weekdays and the subject of content to two separate columns. now we have 48 columns

In [306]:
```
1 simpleLinear(dataset)
2 ridge_regression(dataset,10)
3 lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.6405895162738783
MSE : 0.7416601385526579
RMSE : 0.8611969220524757
r2_score : 0.10350017371882592  10.0

Ridge Model.....................................

The train score for ridge model is 0.12314944020578034
The test score for ridge model is 0.10341063719023047
MAE : 0.6406756249817277
MSE : 0.7417342107077849
r2_score : 0.10341063719023047  10.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001
```

Multicollinearity

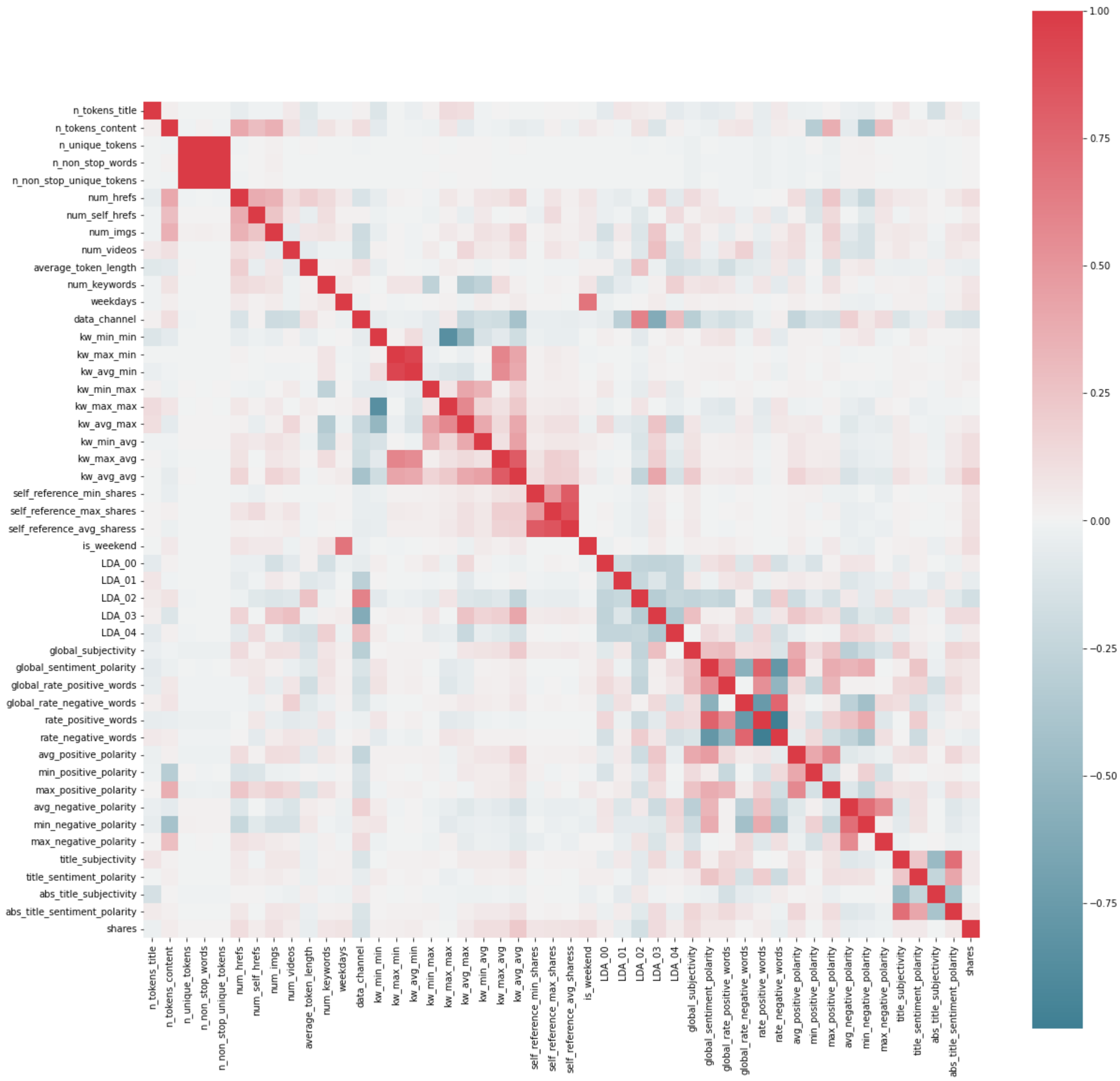
multicollinearity is a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy

In [307]:
```
1 #save the previous dataset
2 dataset_Multicollinearity = dataset.copy()
```

In [308]:
```
f, ax = plt.subplots(figsize=(20, 20))
corr = dataset.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
```

Out[308]: <AxesSubplot:>



In [309]:
```
def correlation(dataset):
    corr = dataset.corr()
    spearmanDataset = {}
    kendallDataset = {}
    feature = dataset.columns
    corrMatrix = abs(corr['shares'][dataset.columns]).values
    for col in dataset.columns:
        spearmanDataset[col] = stats.spearmanr(dataset[col] , dataset['shares']).correlation
        kendallDataset[col] = stats.kendalltau(dataset[col] , dataset['shares']).correlation

    for ele in spearmanDataset:
        spearmanDataset[ele] = abs(spearmanDataset[ele])
    for ele in kendallDataset:
        kendallDataset[ele] = abs(kendallDataset[ele])

    data = {'feature': feature,
                'corrMatrix': corrMatrix,
                'spearmanCorr':  spearmanDataset.values(),
                'kendallCorr':kendallDataset.values(),

            }
    n = len(dataset.columns)
    output = [i for i in range(1, n+1)]
    df = pd.DataFrame(data,index=output)
    print(df)
```

In [310]:
```
correlation(dataset)
```

	feature	corrMatrix	spearmanCorr	kendallCorr
1	n_tokens_title	0.022828	0.042969	0.030741
2	n_tokens_content	0.033145	0.018643	0.012796
3	n_unique_tokens	0.006029	0.039568	0.027031
4	n_non_stop_words	0.006687	0.018603	0.012727
5	n_non_stop_unique_tokens	0.005081	0.067060	0.045443
6	num_hrefs	0.114829	0.104048	0.072051
7	num_self_hrefs	0.035337	0.046500	0.033216
8	num_imgs	0.094330	0.089893	0.067333
9	num_videos	0.033503	0.047208	0.036976
10	average_token_length	0.040344	0.052761	0.035444
11	num_keywords	0.066324	0.072592	0.052202
12	weekdays	0.084397	0.105091	0.076278
13	data_channel	0.132628	0.109956	0.080299
14	kw_min_min	0.024643	0.006278	0.004490
15	kw_max_min	0.032034	0.092275	0.062425
16	kw_avg_min	0.039446	0.095118	0.064154
17	kw_min_max	0.006844	0.055874	0.040534
18	kw_max_max	0.000783	0.032454	0.025510
19	kw_avg_max	0.053282	0.034134	0.022769
20	kw_min_avg	0.107479	0.102372	0.074861
21	kw_max_avg	0.107572	0.224809	0.153848
22	kw_avg_avg	0.221402	0.258609	0.177036
23	self_reference_min_shares	0.076180	0.198995	0.138155
24	self_reference_max_shares	0.080663	0.185388	0.128085
25	self_reference_avg_sharess	0.090758	0.210312	0.145590
26	is_weekend	0.116252	0.153832	0.126623
27	LDA_00	0.033367	0.022897	0.015854
28	LDA_01	0.051785	0.068923	0.046403
29	LDA_02	0.167765	0.160758	0.108276
30	LDA_03	0.123985	0.065158	0.043481
31	LDA_04	0.048193	0.055052	0.037939
32	global_subjectivity	0.133976	0.129058	0.087119
33	global_sentiment_polarity	0.058850	0.090288	0.060818
34	global_rate_positive_words	0.060746	0.084216	0.056624
35	global_rate_negative_words	0.003349	0.015716	0.010467
36	rate_positive_words	0.039992	0.067711	0.045503
37	rate_negative_words	0.040049	0.067730	0.045517
38	avg_positive_polarity	0.069385	0.064307	0.043504
39	min_positive_polarity	0.002893	0.039853	0.029592
40	max_positive_polarity	0.064585	0.069033	0.050777
41	avg_negative_polarity	0.050784	0.026926	0.018171
42	min_negative_polarity	0.036047	0.024426	0.017155
43	max_negative_polarity	0.021819	0.010005	0.007206
44	title_subjectivity	0.052555	0.040277	0.029182
45	title_sentiment_polarity	0.050173	0.057611	0.042258
46	abs_title_subjectivity	0.000098	0.000975	0.000751
47	abs_title_sentiment_polarity	0.058783	0.036283	0.026544
48	shares	1.000000	1.000000	1.000000

In [311]:
```
# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = dataset.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(dataset.values, i)
                          for i in range(len(dataset.columns))]

print(vif_data.sort_values(by=['VIF']))
```

```
                        feature          VIF
0                n_tokens_title      1.088424
47                       shares      1.136131
8                    num_videos      1.257560
44     title_sentiment_polarity      1.329810
6                num_self_hrefs      1.354533
16                   kw_min_max      1.361544
9          average_token_length      1.365337
45       abs_title_subjectivity      1.423067
10                 num_keywords      1.469948
31          global_subjectivity      1.641137
5                       num_hrefs      1.681461
7                       num_imgs      1.712469
38        min_positive_polarity      1.879509
11                      weekdays      1.894946
25                   is_weekend      1.920794
19                   kw_min_avg      2.178924
43           title_subjectivity      2.368668
46  abs_title_sentiment_polarity      2.406149
39        max_positive_polarity      2.437638
42        max_negative_polarity      2.867294
1              n_tokens_content      3.187674
12                 data_channel      3.200613
18                   kw_avg_max      3.235844
13                   kw_min_min      3.813787
33   global_rate_positive_words      3.927468
37        avg_positive_polarity      3.964013
17                   kw_max_max      4.543296
41        min_negative_polarity      4.785519
34   global_rate_negative_words      6.093573
22    self_reference_min_shares      6.601504
40        avg_negative_polarity      6.755807
20                   kw_max_avg      7.025539
32    global_sentiment_polarity      7.463975
23    self_reference_max_shares      8.395084
21                   kw_avg_avg     10.216101
15                   kw_avg_min     10.830847
14                   kw_max_min     11.123887
24   self_reference_avg_sharess     19.102120
35         rate_positive_words    218.885684
36         rate_negative_words    221.841456
27                      LDA_01    699.254823
26                      LDA_00   1081.229887
28                      LDA_02   1300.020706
29                      LDA_03   1317.864808
30                      LDA_04   1447.552607
3             n_non_stop_words   3584.045110
4      n_non_stop_unique_tokens   8430.344074
2               n_unique_tokens  13600.922847
```

In [312]:
```
correlation = dataset.corr().abs()
upper = correlation.where(np.triu(np.ones(correlation.shape),k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column]>0.95)]
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically w
anted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#de
precations)
```

In [313]:
```
to_drop
```

Out[313]:
```
['n_non_stop_words', 'n_non_stop_unique_tokens', 'rate_negative_words']
```

In [314]:
```
to_drop.remove('n_non_stop_unique_tokens')
```

In [315]:
```
dataset.drop('n_non_stop_unique_tokens', inplace=True, axis=1)
```

In [316]:
```
vif_data = pd.DataFrame()
vif_data["feature"] = dataset.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(dataset.values, i)
                          for i in range(len(dataset.columns))]

print(vif_data.sort_values(by=['VIF']))
```

```
                        feature         VIF
0                n_tokens_title    1.088321
46                       shares    1.135804
7                    num_videos    1.246804
8          average_token_length    1.257907
43     title_sentiment_polarity    1.329754
5                num_self_hrefs    1.354262
15                   kw_min_max    1.361454
44       abs_title_subjectivity    1.422433
9                  num_keywords    1.469479
6                      num_imgs    1.534202
30         global_subjectivity    1.640890
4                    num_hrefs    1.659792
37        min_positive_polarity    1.846223
10                    weekdays    1.894857
24                  is_weekend    1.920740
18                   kw_min_avg    2.177784
42           title_subjectivity    2.368638
45  abs_title_sentiment_polarity    2.405452
38        max_positive_polarity    2.420248
41        max_negative_polarity    2.846573
1              n_tokens_content    3.010446
11                 data_channel    3.196860
17                   kw_avg_max    3.234672
12                   kw_min_min    3.813581
32   global_rate_positive_words    3.927342
36        avg_positive_polarity    3.959332
16                   kw_max_max    4.540575
40        min_negative_polarity    4.735513
33   global_rate_negative_words    6.089337
21    self_reference_min_shares    6.600599
39        avg_negative_polarity    6.755578
19                   kw_max_avg    7.025398
31    global_sentiment_polarity    7.461892
22    self_reference_max_shares    8.391662
20                   kw_avg_avg   10.216064
14                   kw_avg_min   10.830827
13                   kw_max_min   11.123850
23   self_reference_avg_sharess   19.096551
34         rate_positive_words  218.858359
35         rate_negative_words  221.805203
26                      LDA_01   693.525393
25                      LDA_00  1071.627230
27                      LDA_02  1288.601902
28                      LDA_03  1308.026412
29                      LDA_04  1435.184953
2              n_unique_tokens  3369.833501
3             n_non_stop_words  3494.909466
```

In [317]:
```
correlation = dataset.corr().abs()
upper = correlation.where(np.triu(np.ones(correlation.shape),k=1).astype(np.bool))
to_drop = [column for column in upper.columns if any(upper[column]>0.95)]
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
```

In [318]:
```
to_drop
```

Out[318]:
```
['n_non_stop_words', 'rate_negative_words']
```

In [319]:
```
dataset.drop('n_non_stop_words', inplace=True, axis=1)
```

In [320]:
```
dataset.drop('rate_negative_words', inplace=True, axis=1)
```

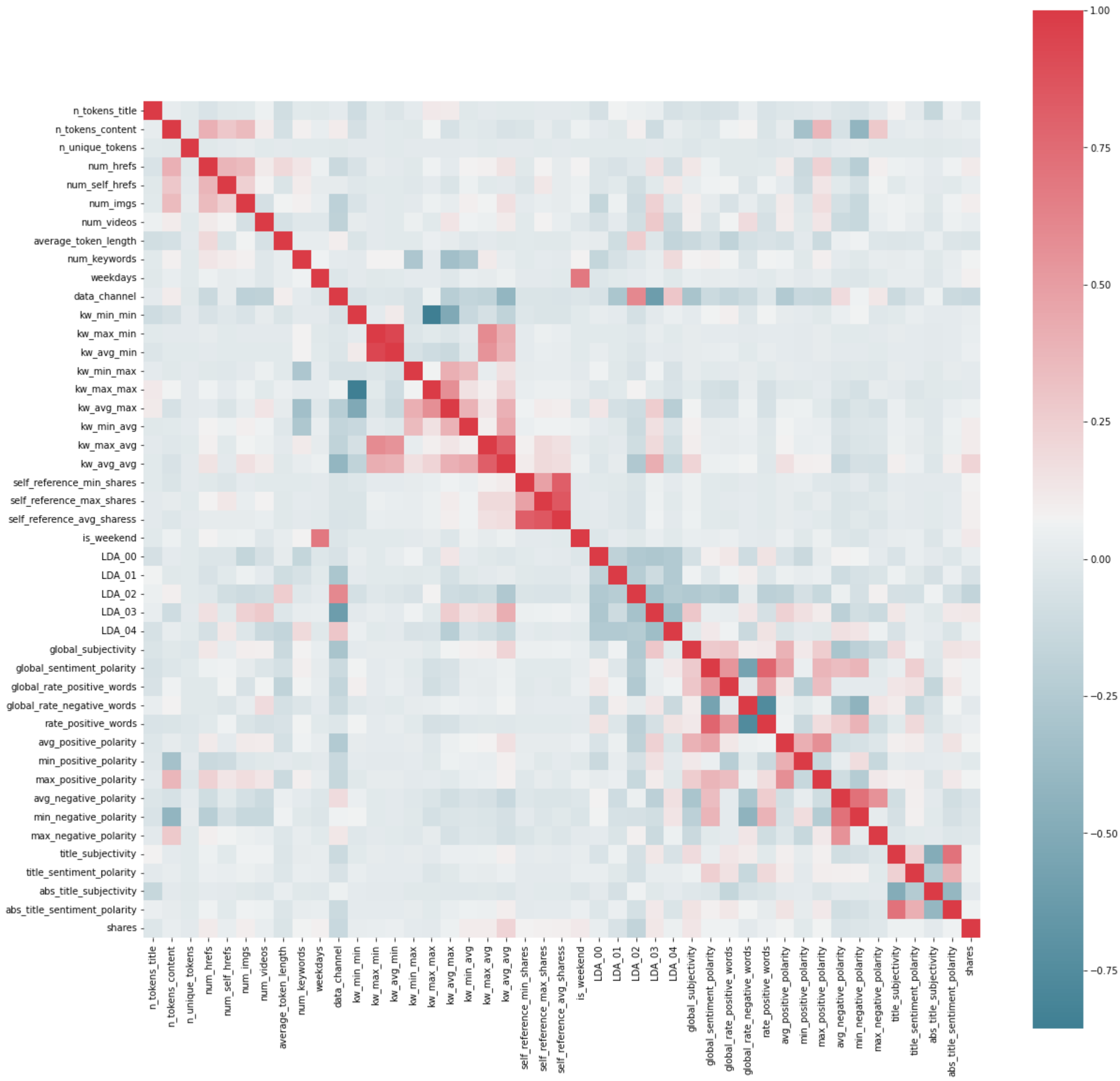
In [321]:
```
vif_data = pd.DataFrame()
vif_data["feature"] = dataset.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(dataset.values, i)
                          for i in range(len(dataset.columns))]

print(vif_data.sort_values(by=['VIF']))
```

	feature	VIF
2	n_unique_tokens	1.039069
6	num_videos	1.357766
41	title_sentiment_polarity	1.424796
14	kw_min_max	1.434464
5	num_imgs	1.931889
23	is_weekend	2.209364
4	num_self_hrefs	2.394481
3	num_hrefs	3.283318
43	abs_title_sentiment_polarity	3.539454
40	title_subjectivity	4.149607
17	kw_min_avg	4.259591
11	kw_min_min	4.363223
1	n_tokens_content	5.262197
35	min_positive_polarity	5.371556
42	abs_title_subjectivity	6.111351
39	max_negative_polarity	6.677017
20	self_reference_min_shares	6.880044
21	self_reference_max_shares	8.940709
9	weekdays	8.975521
10	data_channel	11.622857
12	kw_max_min	12.108401
18	kw_max_avg	12.960684
13	kw_avg_min	13.567908
16	kw_avg_max	15.314481
30	global_sentiment_polarity	19.602320
22	self_reference_avg_sharess	20.475103
32	global_rate_negative_words	21.802498
38	min_negative_polarity	22.186492
8	num_keywords	22.297359
0	n_tokens_title	27.327529
31	global_rate_positive_words	28.281524
36	max_positive_polarity	34.339191
37	avg_negative_polarity	39.653103
29	global_subjectivity	45.501731
25	LDA_01	54.657643
15	kw_max_max	59.090436
19	kw_avg_avg	68.326799
44	shares	75.259117
34	avg_positive_polarity	75.604385
24	LDA_00	85.725636
27	LDA_03	104.172558
26	LDA_02	106.733780
28	LDA_04	113.788445
33	rate_positive_words	185.924480
7	average_token_length	343.919141

In [322]:
```
f, ax = plt.subplots(figsize=(20, 20))
corr = dataset.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

```
`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
```

Out[322]: <AxesSubplot:>



In [323]:
```
dataset
```

Out[323]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
0	12.0	219.0	0.663594	4.0	2.0	1.0	0.0	4.680365	5.0	1	2	0.0	0.0	0.000	
1	9.0	255.0	0.604743	3.0	1.0	1.0	0.0	4.913725	4.0	1	3	0.0	0.0	0.000	
2	9.0	211.0	0.575130	3.0	1.0	1.0	0.0	4.393365	6.0	1	3	0.0	0.0	0.000	
3	9.0	531.0	0.503788	9.0	0.0	1.0	0.0	4.404896	7.0	1	2	0.0	0.0	0.000	
4	13.0	1072.0	0.415646	19.0	19.0	20.0	0.0	4.682836	7.0	1	5	0.0	0.0	0.000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39639	11.0	346.0	0.529052	9.0	7.0	1.0	1.0	4.523121	8.0	3	5	-1.0	671.0	173.125	
39640	12.0	328.0	0.696296	9.0	7.0	3.0	48.0	4.405488	7.0	3	4	-1.0	616.0	184.000	
39641	10.0	442.0	0.516355	24.0	1.0	12.0	1.0	5.076923	8.0	3	0	-1.0	691.0	168.250	
39642	6.0	682.0	0.539493	10.0	1.0	1.0	0.0	4.975073	5.0	3	6	-1.0	0.0	-1.000	
39643	10.0	157.0	0.701987	1.0	1.0	0.0	2.0	4.471338	4.0	3	2	-1.0	97.0	23.500	

38463 rows × 45 columns

In [324]:
```
#dropped three other features in this part 'n_non_stop_words', 'n_non_stop_unique_tokens', 'rate_negative_words'
```

In [325]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.6406746917318741
MSE : 0.7416572118386511
RMSE : 0.8611952228378018
r2_score : 0.10350371145594062  10.0

Ridge Model.....................................

The train score for ridge model is 0.12280559453490081
The test score for ridge model is 0.10345919339787413
MAE : 0.6407935748647433
MSE : 0.7416940409245536
r2_score : 0.10345919339787413  10.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001
```

3.Use various scaling methods and report their effects.

(Exercise 12)
Below are the few ways we can do feature scaling.

1) Min Max Scaler
2) Standard Scaler
3) Max Abs Scaler
4) Robust Scaler

In [326]:
```
dataset_RAW  = dataset.copy()
```

But first we should checkout distributions and try to fix outliers

Methods

using boxplot and histogram and describe for each feature

In [327]:
```
def featureAnalysis(feature,dataset):
    a = dataset.describe()
    b = a[feature].to_frame().T
    fig, ax = plt.subplots(3, 1, figsize=(20, 18))
    sns.histplot(x=dataset[feature], data=dataset, kde=True, element="step",ax=ax[0])
    sns.boxplot(data=dataset , x = feature ,ax=ax[1])
    plt.scatter(dataset[feature] ,dataset['shares'])
    return(b)
```

spearman,kendall and correlation matrix values

In [328]:
```
def correlation(dataset,feat):
    corr = dataset.corr()
    spearmanDataset = {}
    kendallDataset = {}
    feature = dataset.columns
    corrMatrix = abs(corr['shares'][dataset.columns]).values
    for col in dataset.columns:
        spearmanDataset[col] = stats.spearmanr(dataset[col] , dataset['shares']).correlation
        kendallDataset[col] = stats.kendalltau(dataset[col] , dataset['shares']).correlation

    for ele in spearmanDataset:
        spearmanDataset[ele] = abs(spearmanDataset[ele])
    for ele in kendallDataset:
        kendallDataset[ele] = abs(kendallDataset[ele])

    data = {'feature': feature,
                'corrMatrix': corrMatrix,
                'spearmanCorr':  spearmanDataset.values(),
                'kendallCorr':kendallDataset.values(),

        }
    n = len(dataset.columns)
    output = [i for i in range(1, n+1)]
    df = pd.DataFrame(data,index=output)
    if feat =='all' :
        print(df)
    else:
        print(df[df['feature'] == feat])
```

detect outliers of each feature to choose the scaler method(using IQR method for detection)

In [329]:
```
def outlierDetection(dataset):
    dicto = {}
    num_cols = dataset.select_dtypes(['int64','float64']).columns
    for column in num_cols:
        q1 = dataset[column].quantile(0.25)    # First Quartile
        q3 = dataset[column].quantile(0.75)    # Third Quartile
        IQR = q3 - q1                            # Inter Quartile Range
        llimit = q1 - 1.5*IQR                       # Lower Limit
        ulimit = q3 + 1.5*IQR                        # Upper Limit
        outliers = dataset[(dataset[column] < llimit) | (dataset[column] > ulimit)]
        dicto[column] = len(outliers)
    sorted_dicto = sorted(dicto.items(), key=lambda x:x[1])
    return (sorted_dicto)
```

Binnig method(sampling from median)

In [330]:
```
def binningFeature(feature,dataset,bins):
    x_data = np.arange(0, len(dataset))
    y_data = dataset[feature]
    x_bins,bin_edges, misc = binned_statistic(y_data,x_data, statistic="median", bins=bins)
    bin_intervals = pd.IntervalIndex.from_arrays(bin_edges[:-1], bin_edges[1:])
    def set_to_median(x, bin_intervals):
        for interval in bin_intervals:
            if x in interval:
                return interval.mid
        for interval in bin_intervals:
            if x in interval:
                return interval.mid
    dataset[feature] = dataset[feature].apply(lambda x: set_to_median(x, bin_intervals))

```

Checking our dataset's features distribution and outliers

In [331]:
```
correlation(dataset_RAW,'all')
```

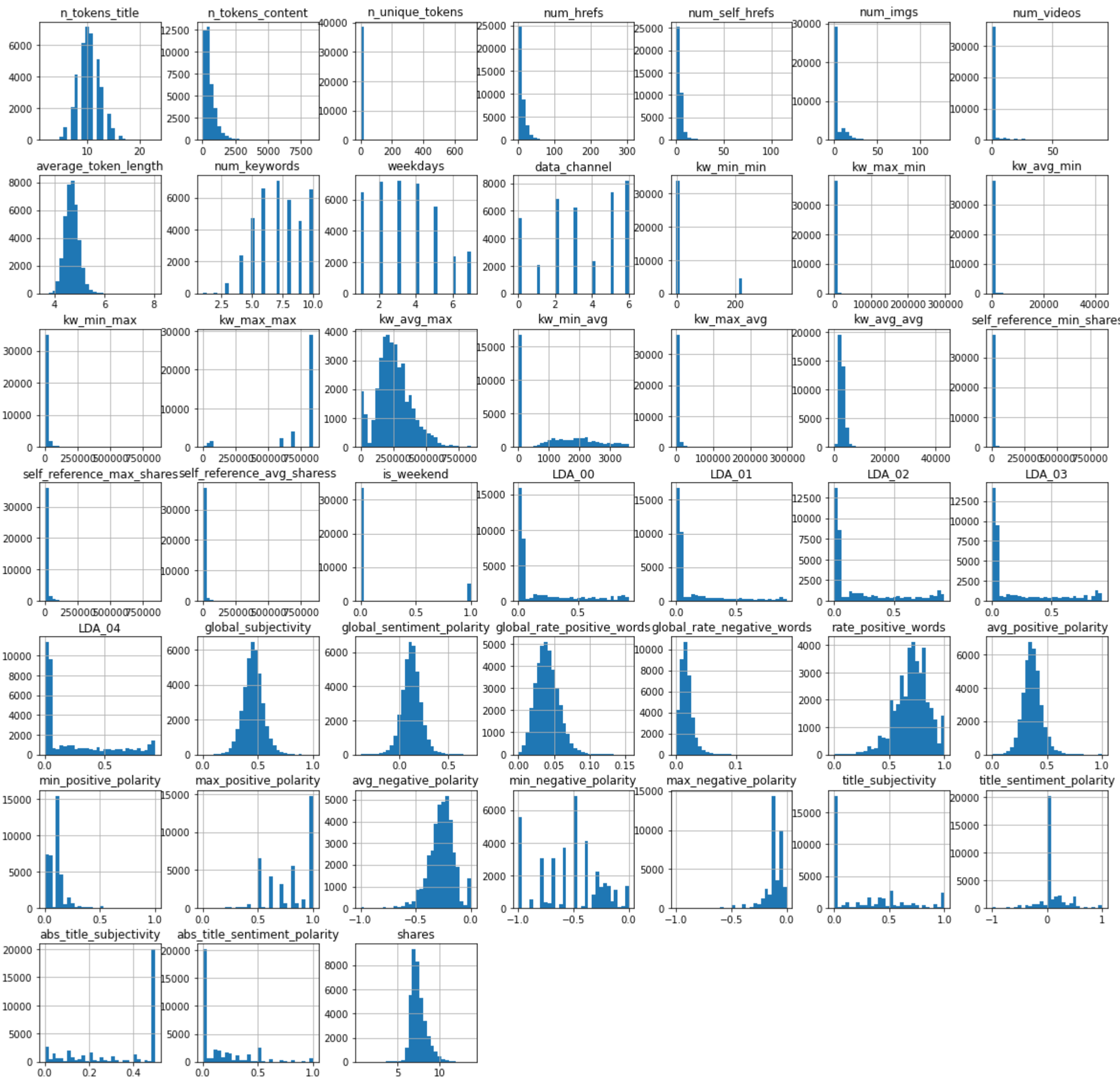
	feature	corrMatrix	spearmanCorr	kendallCorr
1	n_tokens_title	0.022828	0.042969	0.030741
2	n_tokens_content	0.033145	0.018643	0.012796
3	n_unique_tokens	0.006029	0.039568	0.027031
4	num_hrefs	0.114829	0.104048	0.072051
5	num_self_hrefs	0.035337	0.046500	0.033216
6	num_imgs	0.094330	0.089893	0.067333
7	num_videos	0.033503	0.047208	0.036976
8	average_token_length	0.040344	0.052761	0.035444
9	num_keywords	0.066324	0.072592	0.052202
10	weekdays	0.084397	0.105091	0.076278
11	data_channel	0.132628	0.109956	0.080299
12	kw_min_min	0.024643	0.006278	0.004490
13	kw_max_min	0.032034	0.092275	0.062425
14	kw_avg_min	0.039446	0.095118	0.064154
15	kw_min_max	0.006844	0.055874	0.040534
16	kw_max_max	0.000783	0.032454	0.025510
17	kw_avg_max	0.053282	0.034134	0.022769
18	kw_min_avg	0.107479	0.102372	0.074861
19	kw_max_avg	0.107572	0.224809	0.153848
20	kw_avg_avg	0.221402	0.258609	0.177036
21	self_reference_min_shares	0.076180	0.198995	0.138155
22	self_reference_max_shares	0.080663	0.185388	0.128085
23	self_reference_avg_sharess	0.090758	0.210312	0.145590
24	is_weekend	0.116252	0.153832	0.126623
25	LDA_00	0.033367	0.022897	0.015854
26	LDA_01	0.051785	0.068923	0.046403
27	LDA_02	0.167765	0.160758	0.108276
28	LDA_03	0.123985	0.065158	0.043481
29	LDA_04	0.048193	0.055052	0.037939
30	global_subjectivity	0.133976	0.129058	0.087119
31	global_sentiment_polarity	0.058850	0.090288	0.060818
32	global_rate_positive_words	0.060746	0.084216	0.056624
33	global_rate_negative_words	0.003349	0.015716	0.010467
34	rate_positive_words	0.039992	0.067711	0.045503
35	avg_positive_polarity	0.069385	0.064307	0.043504
36	min_positive_polarity	0.002893	0.039853	0.029592
37	max_positive_polarity	0.064585	0.069033	0.050777
38	avg_negative_polarity	0.050784	0.026926	0.018171
39	min_negative_polarity	0.036047	0.024426	0.017155
40	max_negative_polarity	0.021819	0.010005	0.007206
41	title_subjectivity	0.052555	0.040277	0.029182
42	title_sentiment_polarity	0.050173	0.057611	0.042258
43	abs_title_subjectivity	0.000098	0.000975	0.000751
44	abs_title_sentiment_polarity	0.058783	0.036283	0.026544
45	shares	1.000000	1.000000	1.000000

In [332]:
```
outlierDetection(dataset)
```

Out[332]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('abs_title_subjectivity', 0),
 ('num_keywords', 50),
 ('n_tokens_title', 155),
 ('n_unique_tokens', 490),
 ('rate_positive_words', 519),
 ('global_rate_positive_words', 520),
 ('average_token_length', 552),
 ('global_subjectivity', 864),
 ('kw_avg_max', 879),
 ('avg_negative_polarity', 879),
 ('global_sentiment_polarity', 944),
 ('avg_positive_polarity', 1042),
 ('global_rate_negative_words', 1358),
 ('abs_title_sentiment_polarity', 1589),
 ('shares', 1599),
 ('kw_avg_avg', 1642),
 ('n_tokens_content', 1875),
 ('kw_avg_min', 2009),
 ('num_self_hrefs', 2090),
 ('kw_max_avg', 2352),
 ('max_negative_polarity', 2454),
 ('num_hrefs', 2636),
 ('num_videos', 2939),
 ('LDA_03', 3029),
 ('min_positive_polarity', 3147),
 ('LDA_02', 3441),
 ('kw_max_min', 3566),
 ('self_reference_max_shares', 4094),
 ('self_reference_avg_sharess', 4175),
 ('kw_min_min', 4638),
 ('kw_min_max', 4827),
 ('self_reference_min_shares', 4884),
 ('is_weekend', 5026),
 ('LDA_00', 5071),
 ('LDA_01', 5682),
 ('num_imgs', 7466),
 ('title_sentiment_polarity', 7918),
 ('kw_max_max', 9365)]
```

In [333]:
```
1 dataset.hist(bins=30,figsize=(20,20))
```

Out[333]:
```
array([[<AxesSubplot:title={'center':'n_tokens_title'}>,
        <AxesSubplot:title={'center':'n_tokens_content'}>,
        <AxesSubplot:title={'center':'n_unique_tokens'}>,
        <AxesSubplot:title={'center':'num_hrefs'}>,
        <AxesSubplot:title={'center':'num_self_hrefs'}>,
        <AxesSubplot:title={'center':'num_imgs'}>,
        <AxesSubplot:title={'center':'num_videos'}>],
       [<AxesSubplot:title={'center':'average_token_length'}>,
        <AxesSubplot:title={'center':'num_keywords'}>,
        <AxesSubplot:title={'center':'weekdays'}>,
        <AxesSubplot:title={'center':'data_channel'}>,
        <AxesSubplot:title={'center':'kw_min_min'}>,
        <AxesSubplot:title={'center':'kw_max_min'}>,
        <AxesSubplot:title={'center':'kw_avg_min'}>],
       [<AxesSubplot:title={'center':'kw_min_max'}>,
        <AxesSubplot:title={'center':'kw_max_max'}>,
        <AxesSubplot:title={'center':'kw_avg_max'}>,
        <AxesSubplot:title={'center':'kw_min_avg'}>,
        <AxesSubplot:title={'center':'kw_max_avg'}>,
        <AxesSubplot:title={'center':'kw_avg_avg'}>,
        <AxesSubplot:title={'center':'self_reference_min_shares'}>],
       [<AxesSubplot:title={'center':'self_reference_max_shares'}>,
        <AxesSubplot:title={'center':'self_reference_avg_sharess'}>,
        <AxesSubplot:title={'center':'is_weekend'}>,
        <AxesSubplot:title={'center':'LDA_00'}>,
        <AxesSubplot:title={'center':'LDA_01'}>,
        <AxesSubplot:title={'center':'LDA_02'}>,
        <AxesSubplot:title={'center':'LDA_03'}>],
       [<AxesSubplot:title={'center':'LDA_04'}>,
        <AxesSubplot:title={'center':'global_subjectivity'}>,
        <AxesSubplot:title={'center':'global_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'global_rate_positive_words'}>,
        <AxesSubplot:title={'center':'global_rate_negative_words'}>,
        <AxesSubplot:title={'center':'rate_positive_words'}>,
        <AxesSubplot:title={'center':'avg_positive_polarity'}>],
       [<AxesSubplot:title={'center':'min_positive_polarity'}>,
        <AxesSubplot:title={'center':'max_positive_polarity'}>,
        <AxesSubplot:title={'center':'avg_negative_polarity'}>,
        <AxesSubplot:title={'center':'min_negative_polarity'}>,
        <AxesSubplot:title={'center':'max_negative_polarity'}>,
        <AxesSubplot:title={'center':'title_subjectivity'}>,
        <AxesSubplot:title={'center':'title_sentiment_polarity'}>],
       [<AxesSubplot:title={'center':'abs_title_subjectivity'}>,
        <AxesSubplot:title={'center':'abs_title_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'shares'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



In [334]:
```
1 dataset
```

Out[334]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
0	12.0	219.0	0.663594	4.0	2.0	1.0	0.0	4.680365	5.0	1	2	0.0	0.0	0.000	
1	9.0	255.0	0.604743	3.0	1.0	1.0	0.0	4.913725	4.0	1	3	0.0	0.0	0.000	
2	9.0	211.0	0.575130	3.0	1.0	1.0	0.0	4.393365	6.0	1	3	0.0	0.0	0.000	
3	9.0	531.0	0.503788	9.0	0.0	1.0	0.0	4.404896	7.0	1	2	0.0	0.0	0.000	
4	13.0	1072.0	0.415646	19.0	19.0	20.0	0.0	4.682836	7.0	1	5	0.0	0.0	0.000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39639	11.0	346.0	0.529052	9.0	7.0	1.0	1.0	4.523121	8.0	3	5	-1.0	671.0	173.125	
39640	12.0	328.0	0.696296	9.0	7.0	3.0	48.0	4.405488	7.0	3	4	-1.0	616.0	184.000	
39641	10.0	442.0	0.516355	24.0	1.0	12.0	1.0	5.076923	8.0	3	0	-1.0	691.0	168.250	
39642	6.0	682.0	0.539493	10.0	1.0	1.0	0.0	4.975073	5.0	3	6	-1.0	0.0	-1.000	
39643	10.0	157.0	0.701987	1.0	1.0	0.0	2.0	4.471338	4.0	3	2	-1.0	97.0	23.500	

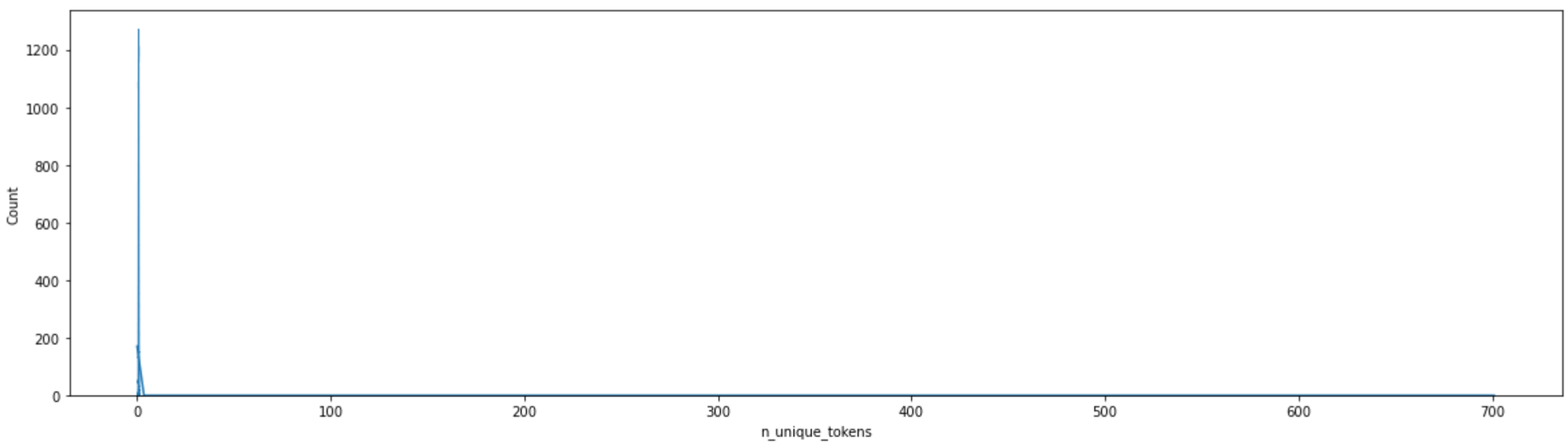
38463 rows × 45 columns

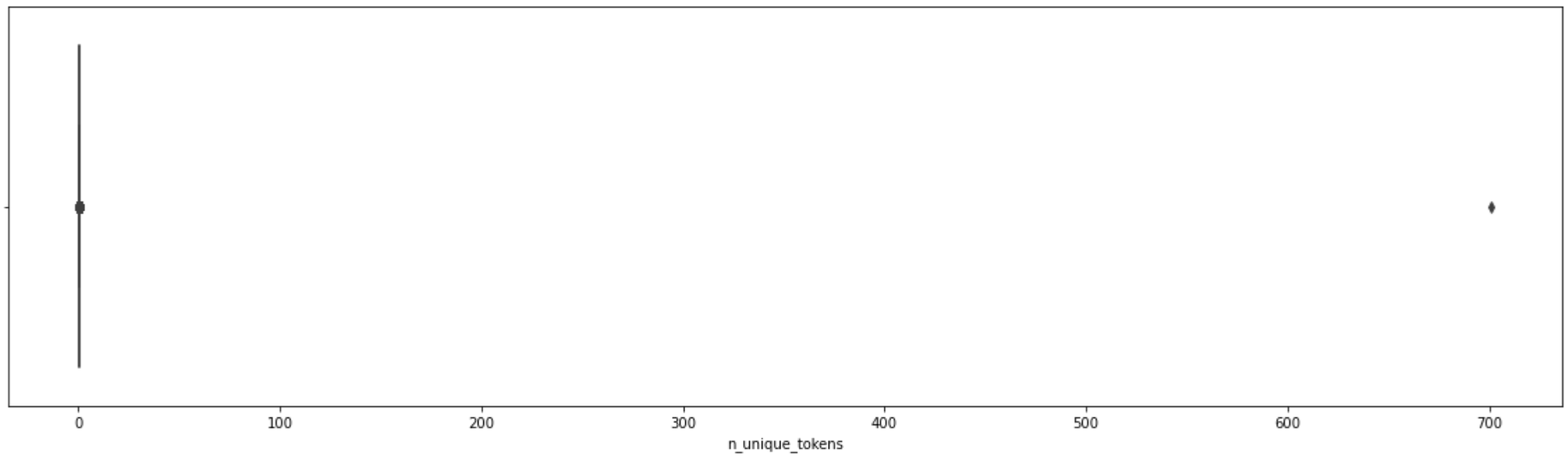
n_unique_tokens

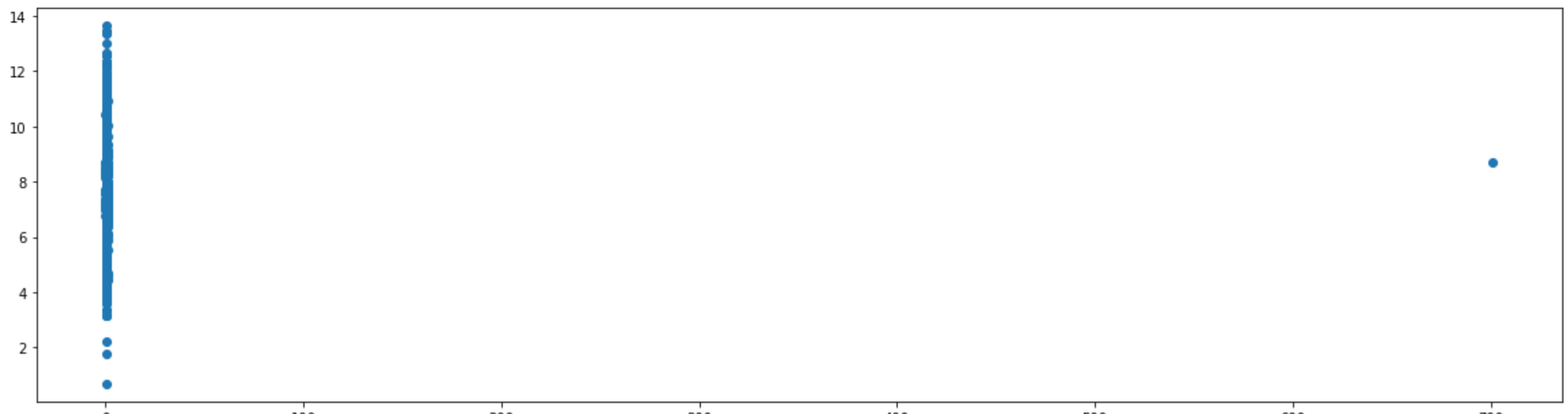
In [335]:
```
1 featureAnalysis('n_unique_tokens',dataset)
```

Out[335]:

	count	mean	std	min	25%	50%	75%	max
n_unique_tokens	38463.0	0.565049	3.573022	0.114964	0.477419	0.542986	0.611111	701.0







In [336]:
```
1 dataset[dataset['n_unique_tokens'] > 300]
```

Out[336]:

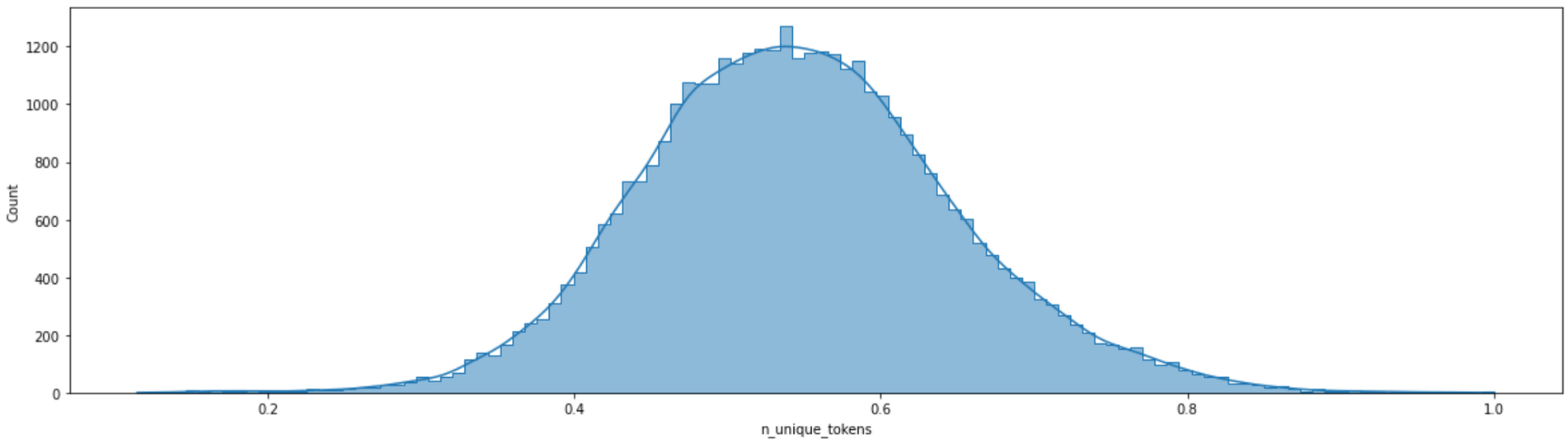
	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
31037	9.0	1570.0	701.0	11.0	10.0	51.0	0.0	4.696178	7.0	2	2	-1.0	778.0	143.714286	

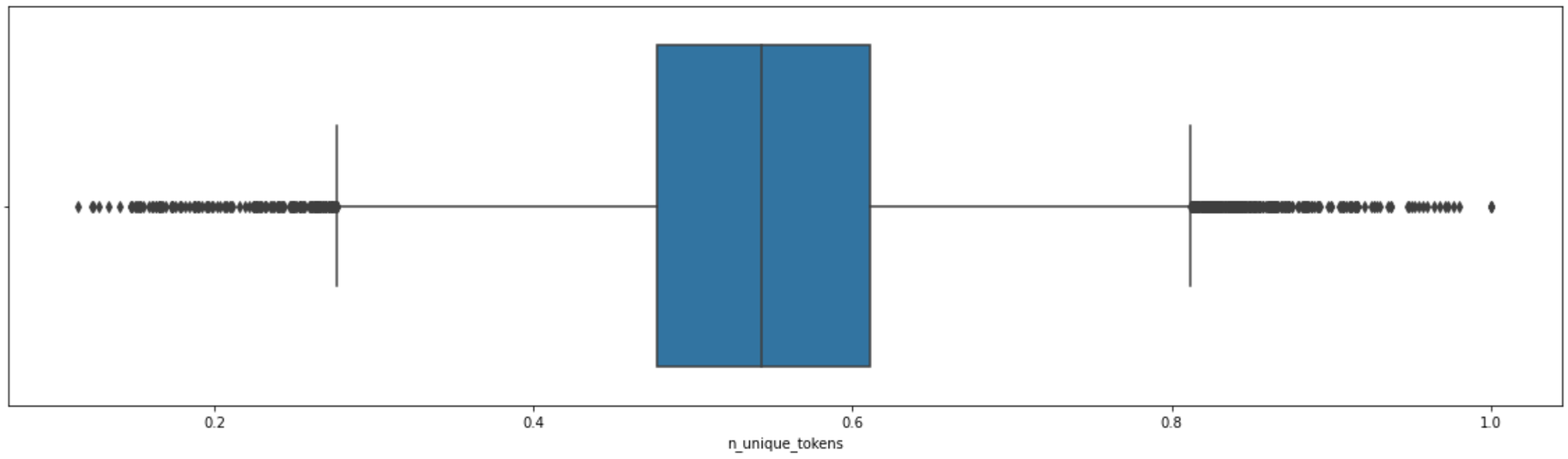
In [337]:
```
1 dataset = dataset[dataset['n_unique_tokens'] < 300]
```

In [338]:
```
1 featureAnalysis('n_unique_tokens',dataset)
```

Out[338]:

	count	mean	std	min	25%	50%	75%	max
n_unique_tokens	38462.0	0.546837	0.102314	0.114964	0.477419	0.542982	0.611111	1.0





In [339]:
```
1 correlation(dataset,'n_unique_tokens')
```

	feature	corrMatrix	spearmanCorr	kendallCorr
3	n_unique_tokens	0.022887	0.039634	0.027074

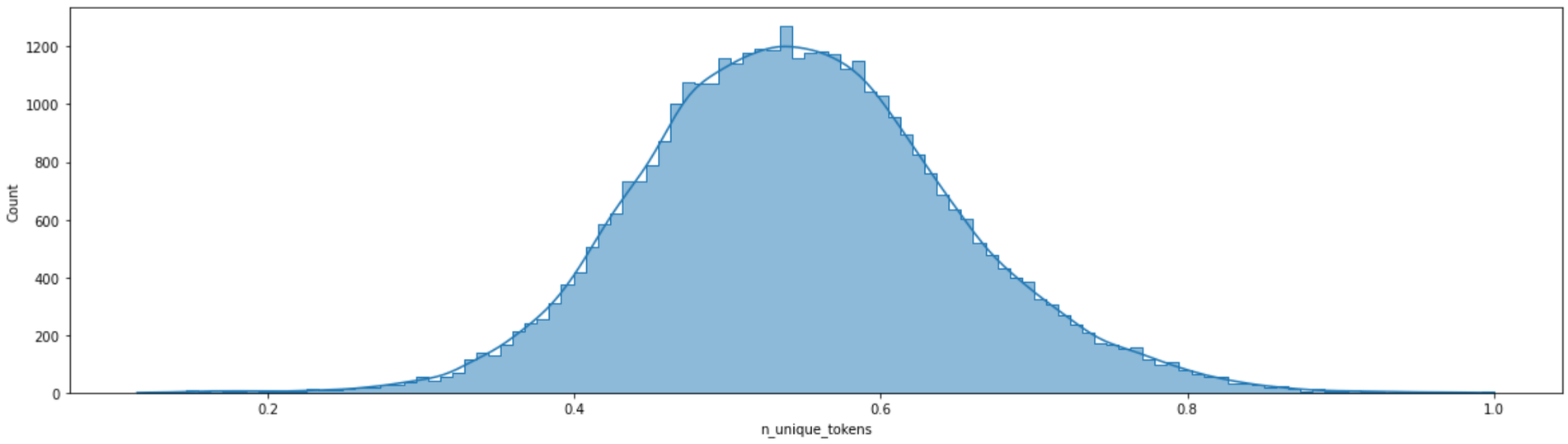
In [340]:
```
1 correlation(dataset_RAW,'n_unique_tokens')
```

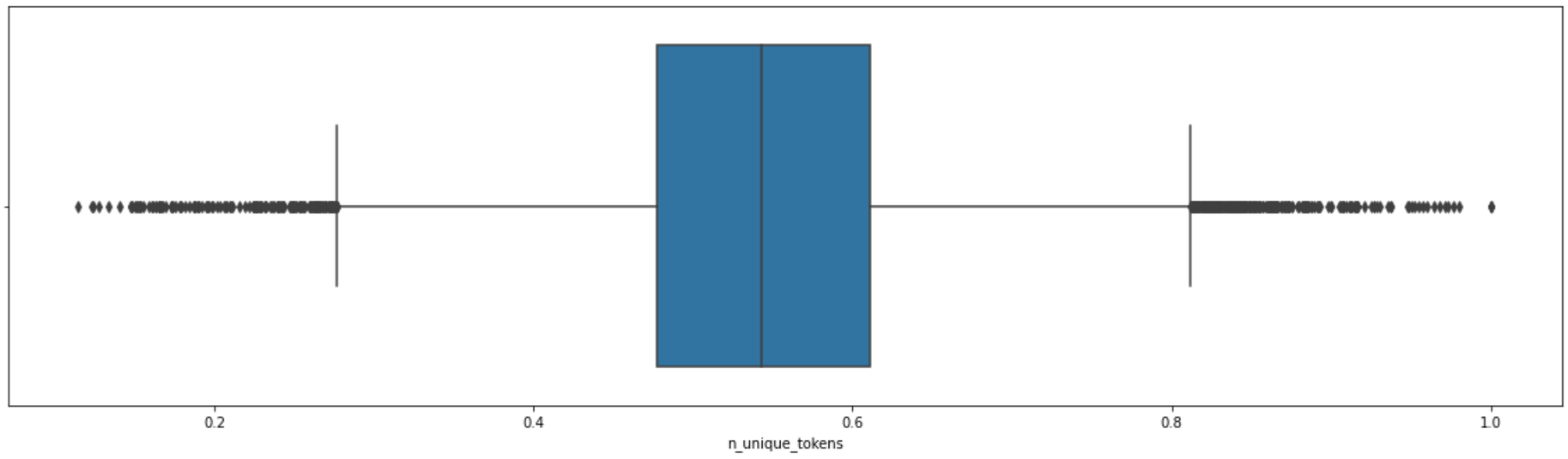
	feature	corrMatrix	spearmanCorr	kendallCorr
3	n_unique_tokens	0.006029	0.039568	0.027031

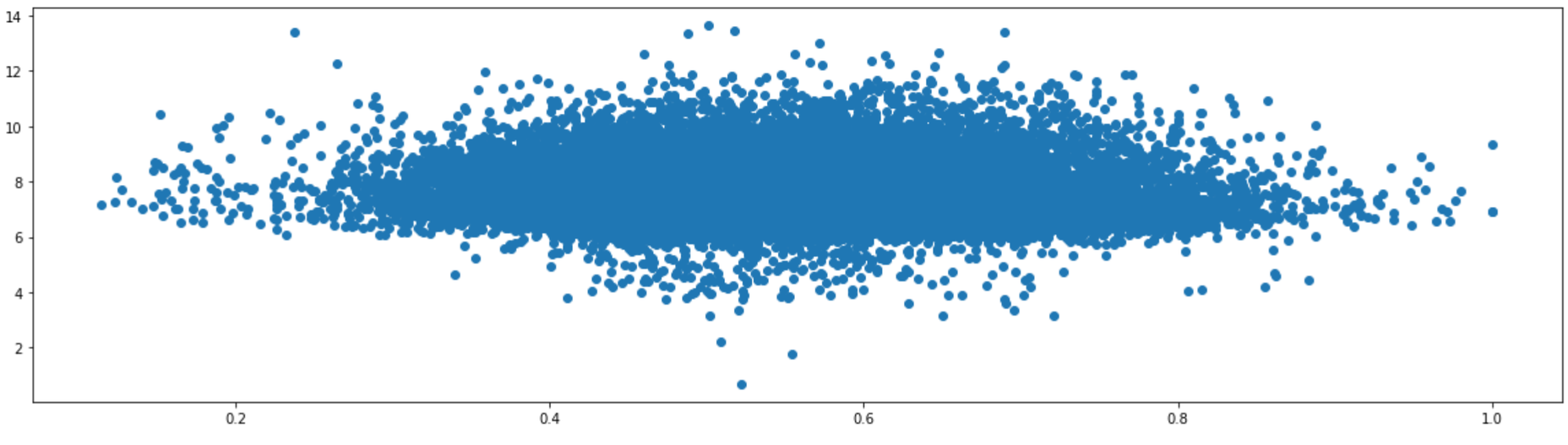
In [341]:
```
1 featureAnalysis('n_unique_tokens',dataset)
```

Out[341]:

	count	mean	std	min	25%	50%	75%	max
n_unique_tokens	38462.0	0.546837	0.102314	0.114964	0.477419	0.542982	0.611111	1.0





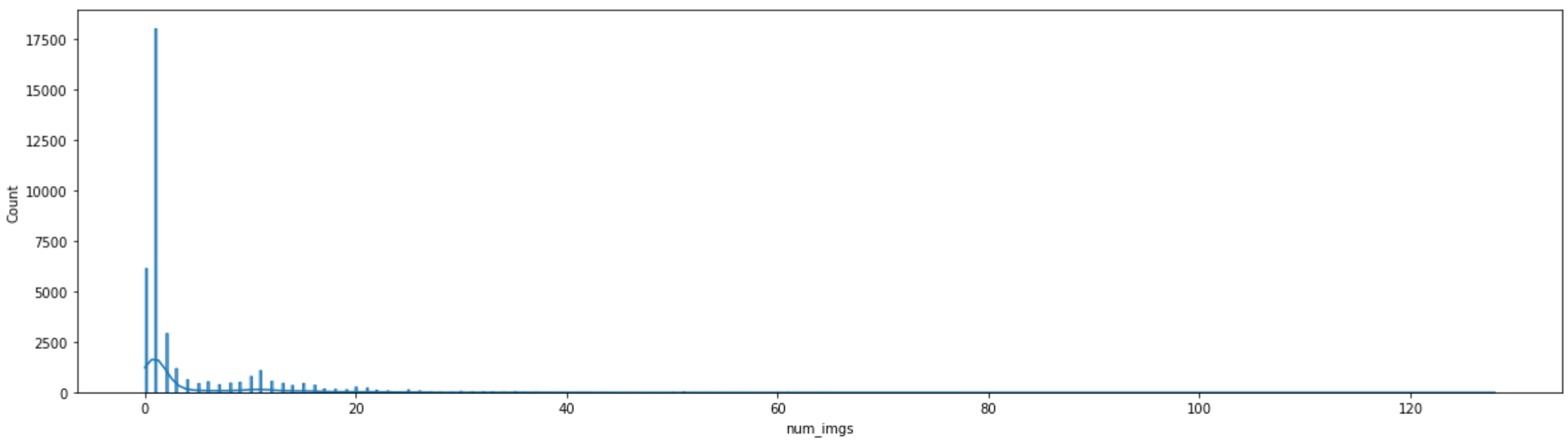


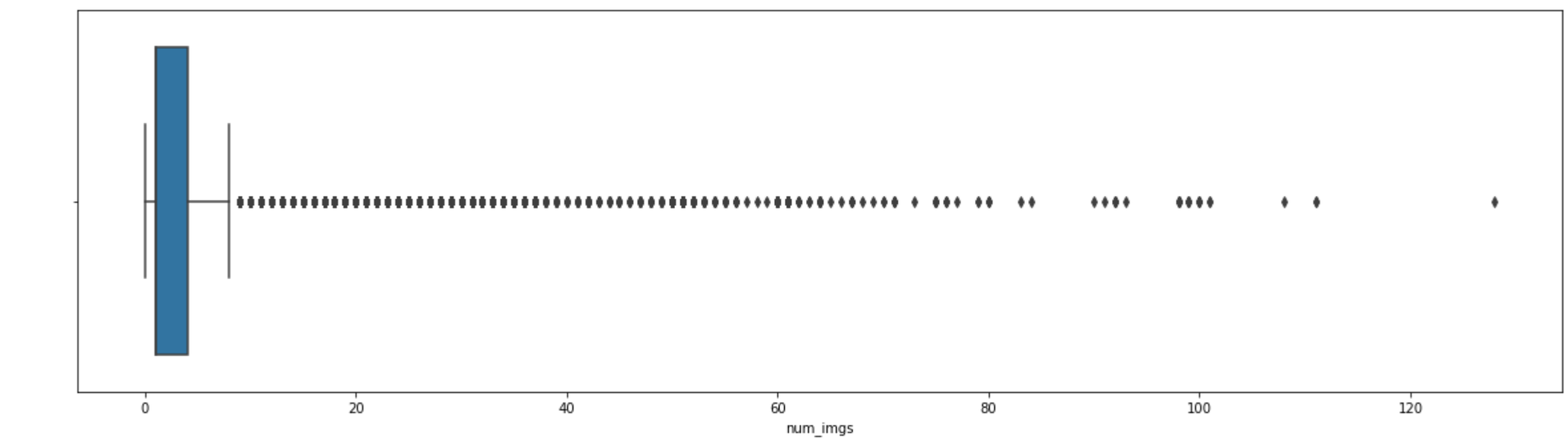
num_imgs/num_videos

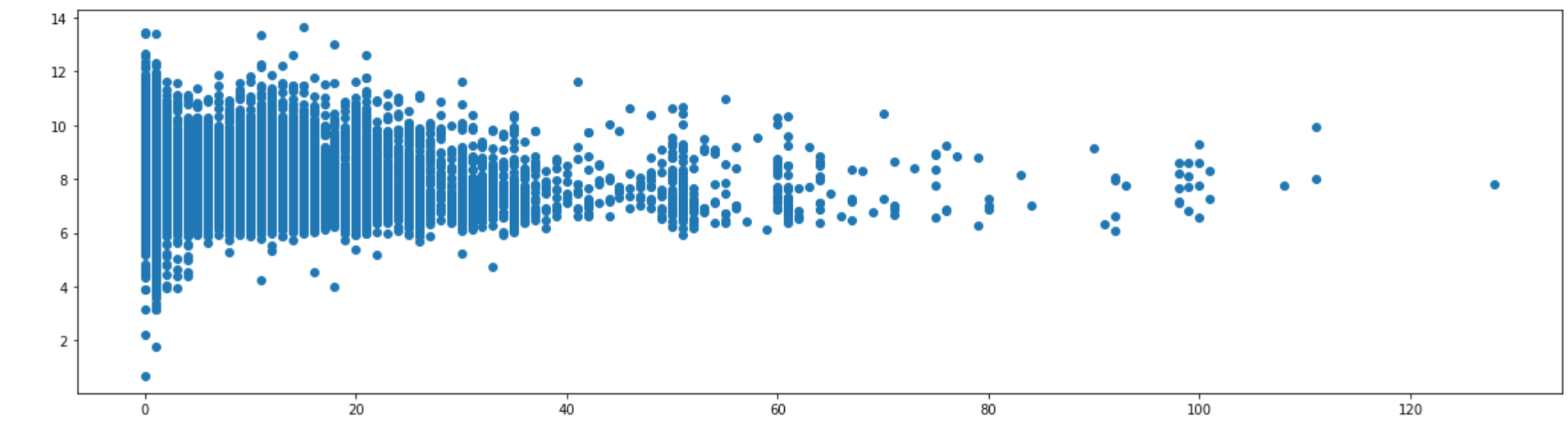
In [342]:
```
1 featureAnalysis('num_imgs',dataset)
```

Out[342]:

	count	mean	std	min	25%	50%	75%	max
num_imgs	38462.0	4.561853	8.292093	0.0	1.0	1.0	4.0	128.0



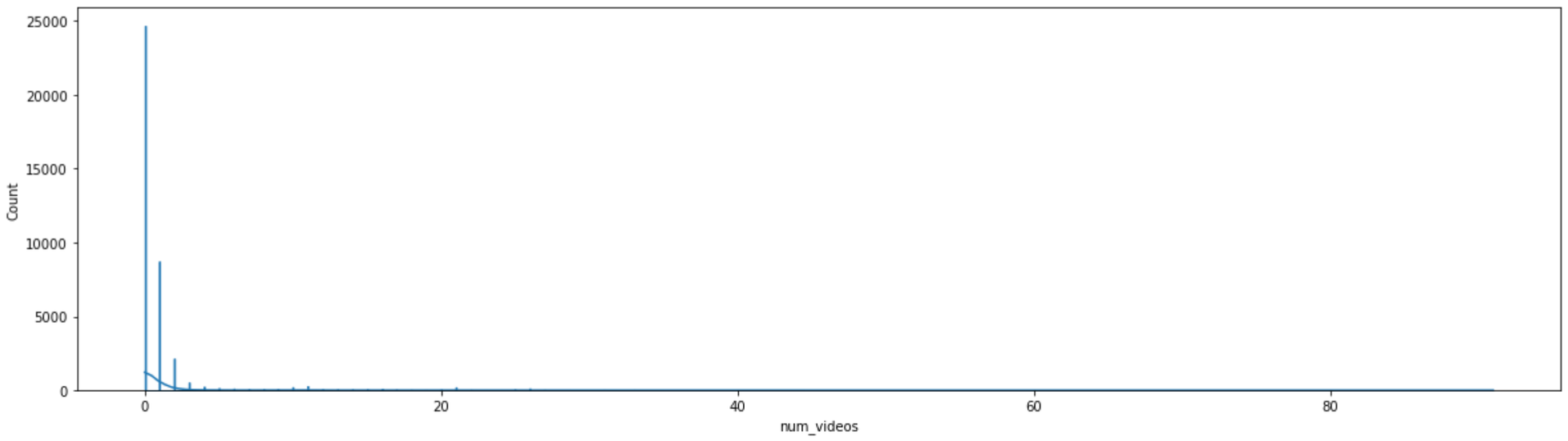


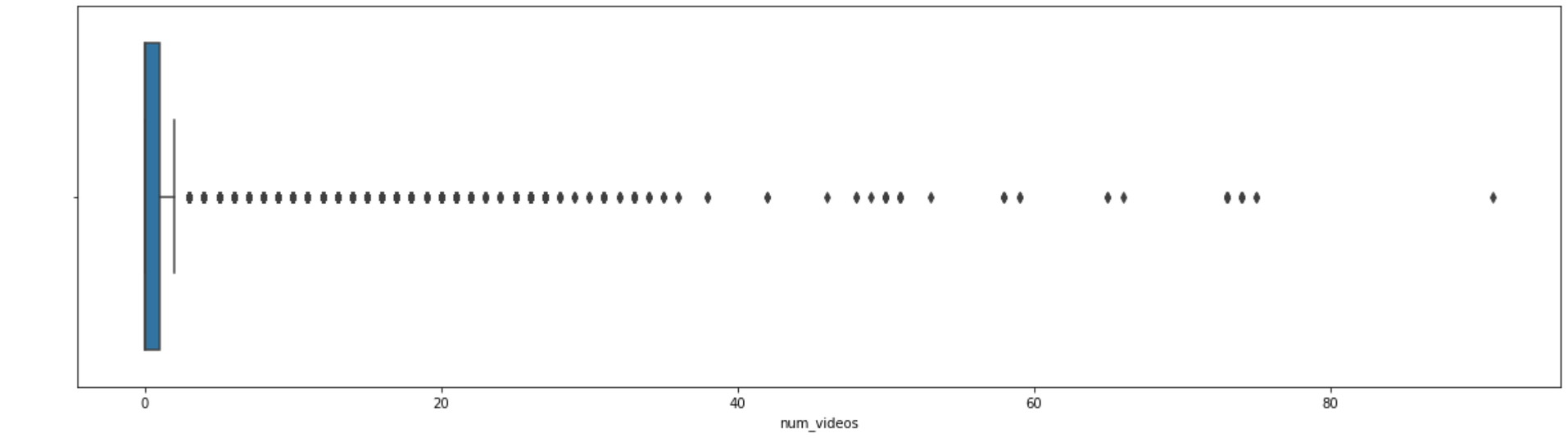


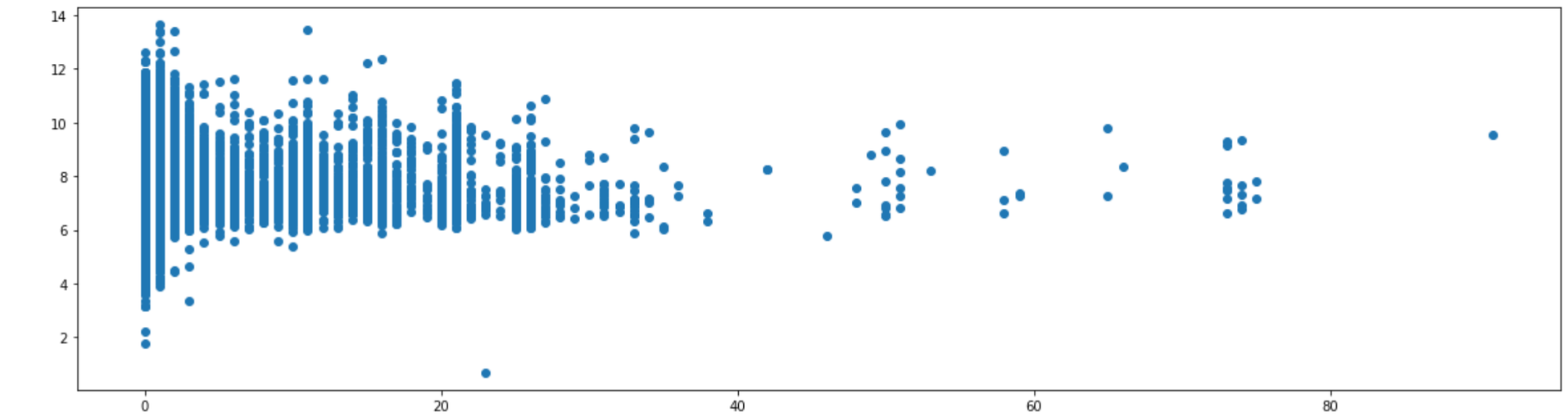
In [343]:
```
1 featureAnalysis('num_videos',dataset)
```

Out[343]:

	count	mean	std	min	25%	50%	75%	max
num_videos	38462.0	1.263819	4.164945	0.0	0.0	0.0	1.0	91.0







In [344]:
```
1 dataset[dataset['num_videos']==0]
```

Out[344]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
0	12.0	219.0	0.663594	4.0	2.0	1.0	0.0	4.680365	5.0	1	2	0.0	0.0	0.000000	
1	9.0	255.0	0.604743	3.0	1.0	1.0	0.0	4.913725	4.0	1	3	0.0	0.0	0.000000	
2	9.0	211.0	0.575130	3.0	1.0	1.0	0.0	4.393365	6.0	1	3	0.0	0.0	0.000000	
3	9.0	531.0	0.503788	9.0	0.0	1.0	0.0	4.404896	7.0	1	2	0.0	0.0	0.000000	
4	13.0	1072.0	0.415646	19.0	19.0	20.0	0.0	4.682836	7.0	1	5	0.0	0.0	0.000000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39635	13.0	478.0	0.514039	18.0	2.0	2.0	0.0	4.891213	6.0	2	3	-1.0	530.0	102.863333	
39636	8.0	2509.0	0.348878	23.0	1.0	10.0	0.0	4.569550	10.0	3	1	-1.0	646.0	152.300000	
39637	13.0	1629.0	0.425711	15.0	12.0	6.0	0.0	4.552486	8.0	3	2	-1.0	1100.0	354.000000	
39638	11.0	223.0	0.653153	5.0	3.0	1.0	0.0	4.923767	6.0	3	3	-1.0	459.0	91.000000	
39642	6.0	682.0	0.539493	10.0	1.0	1.0	0.0	4.975073	5.0	3	6	-1.0	0.0	-1.000000	

24660 rows × 45 columns

In [345]:
```
1 24660 / 39642
```

Out[345]: 0.6220675041622522

In [346]:
```
1 dataset[dataset['num_imgs']==0]
```

Out[346]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
5	10.0	370.0	0.559889	2.0	2.0	0.0	0.0	4.359459	9.0	1	5	0.0	0.0	0.000000	
8	11.0	97.0	0.670103	2.0	0.0	0.0	0.0	4.855670	7.0	1	5	0.0	0.0	0.000000	
13	9.0	285.0	0.744186	4.0	2.0	0.0	21.0	4.343860	6.0	1	0	0.0	0.0	0.000000	
28	10.0	243.0	0.619247	1.0	1.0	0.0	0.0	4.382716	10.0	1	1	0.0	0.0	0.000000	
37	8.0	257.0	0.568093	9.0	7.0	0.0	1.0	4.638132	9.0	1	4	0.0	0.0	0.000000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39447	8.0	304.0	0.609589	9.0	1.0	0.0	0.0	4.766447	6.0	5	6	-1.0	712.0	184.500000	
39560	9.0	261.0	0.620553	4.0	3.0	0.0	1.0	4.295019	6.0	1	5	-1.0	49.0	7.333333	
39595	12.0	334.0	0.573171	2.0	1.0	0.0	1.0	4.329341	6.0	2	6	-1.0	0.0	-1.000000	
39629	13.0	209.0	0.656863	4.0	1.0	0.0	2.0	4.856459	5.0	2	2	-1.0	423.0	100.800000	
39643	10.0	157.0	0.701987	1.0	1.0	0.0	2.0	4.471338	4.0	3	2	-1.0	97.0	23.500000	

6170 rows × 45 columns

In [347]:
```
6170 /39643
```

Out[347]: 0.1556390787780945

In [348]:
```
dataset[dataset['num_imgs']+dataset['num_videos']==0]
```

Out[348]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
5	10.0	370.0	0.559889	2.0	2.0	0.0	0.0	4.359459	9.0	1	5	0.0	0.0	0.000000	
8	11.0	97.0	0.670103	2.0	0.0	0.0	0.0	4.855670	7.0	1	5	0.0	0.0	0.000000	
28	10.0	243.0	0.619247	1.0	1.0	0.0	0.0	4.382716	10.0	1	1	0.0	0.0	0.000000	
57	8.0	130.0	0.821705	7.0	4.0	0.0	0.0	4.546154	9.0	1	5	0.0	0.0	0.000000	
79	12.0	288.0	0.589474	5.0	2.0	0.0	0.0	4.381944	6.0	2	6	217.0	504.0	421.750000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39271	8.0	129.0	0.703125	3.0	1.0	0.0	0.0	4.178295	10.0	1	0	-1.0	569.0	172.600000	
39329	9.0	358.0	0.548023	3.0	1.0	0.0	0.0	4.466480	6.0	2	5	-1.0	801.0	269.666667	
39412	13.0	99.0	0.762887	14.0	1.0	0.0	0.0	4.434343	8.0	4	0	-1.0	432.0	102.625000	
39420	10.0	199.0	0.682051	4.0	1.0	0.0	0.0	4.221106	6.0	4	6	-1.0	129.0	38.600000	
39447	8.0	304.0	0.609589	9.0	1.0	0.0	0.0	4.766447	6.0	5	6	-1.0	712.0	184.500000	

2036 rows × 45 columns

In [349]:
```
2036 /39447
```

Out[349]: 0.051613557431490355

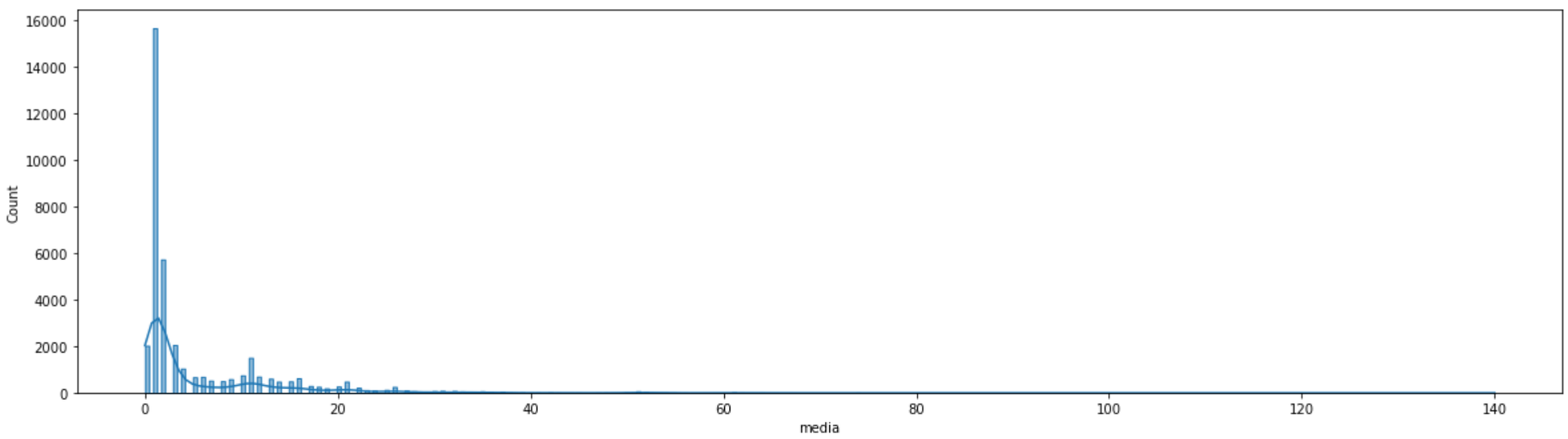
In [350]:
```
dataset['media'] = dataset['num_imgs']+dataset['num_videos']
```

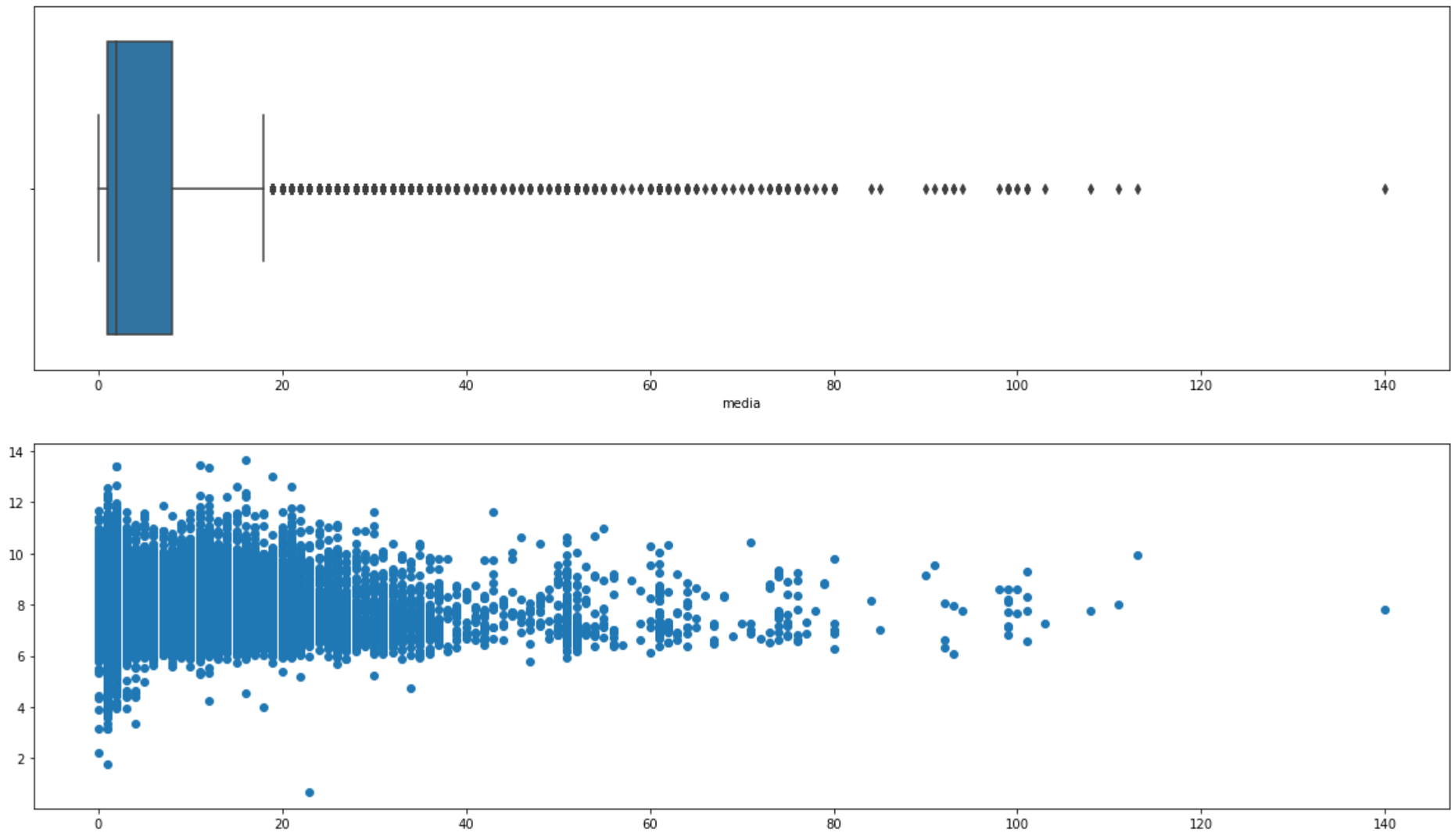
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [351]:
```
featureAnalysis('media',dataset)
```

Out[351]:

	count	mean	std	min	25%	50%	75%	max
media	38462.0	5.825672	9.028125	0.0	1.0	2.0	8.0	140.0





In [352]:
```
dataset['media'] = np.log1p(dataset.media)
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [353]:
```
correlation(dataset,'media')
```

```
   feature  corrMatrix  spearmanCorr  kendallCorr
46   media    0.134062       0.12166      0.08898
```

In [354]:
```
1 correlation(dataset,'num_imgs')
```

```
    feature  corrMatrix  spearmanCorr  kendallCorr
6  num_imgs     0.09418      0.089835     0.067289
```

In [355]:
```
1 correlation(dataset,'num_videos')
```

```
      feature  corrMatrix  spearmanCorr  kendallCorr
7  num_videos    0.033514      0.047234     0.036996
```

change video and media and test correlation and outliers

In [356]:
```
1 dataset2 = dataset.copy()
```

In [357]:
```
1 dataset['num_imgs'] = np.sqrt(dataset.num_imgs)
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [358]:
```
1 dataset['num_videos'] = np.sqrt(dataset.num_videos)
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [359]:
```
1 correlation(dataset,'num_imgs')
```

```
    feature  corrMatrix  spearmanCorr  kendallCorr
6  num_imgs    0.099305      0.089835     0.067289
```

In [360]:
```
1 correlation(dataset,'num_videos')
```

```
      feature  corrMatrix  spearmanCorr  kendallCorr
7  num_videos    0.057621      0.047234     0.036996
```

n_tokens_content / average_token_length

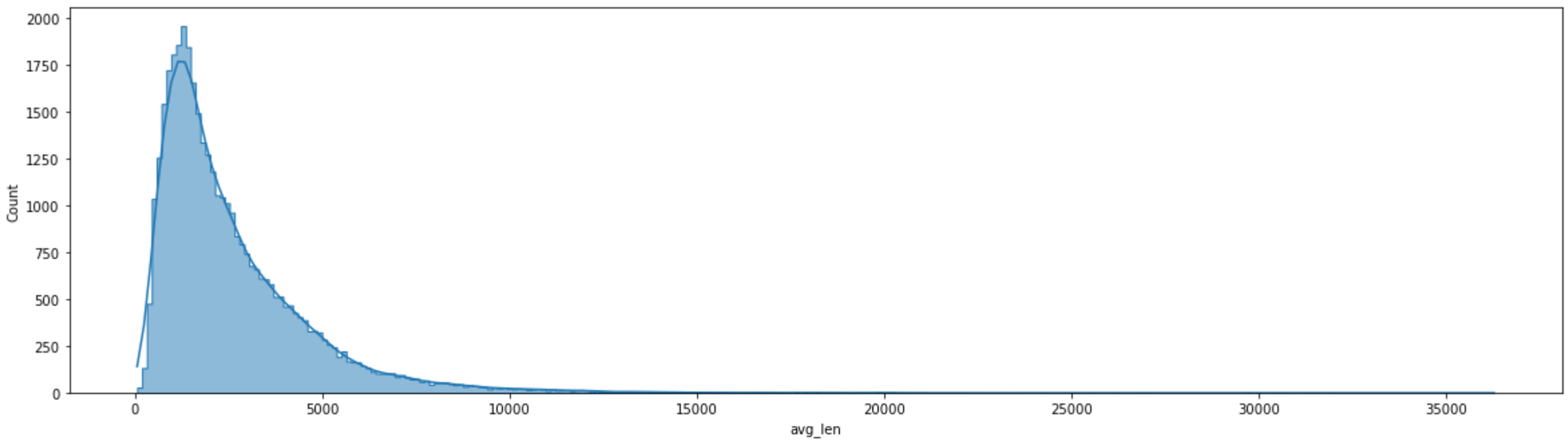
In [361]:
```
1 dataset['avg_len'] = dataset["n_tokens_content"] * dataset["average_token_length"]
```

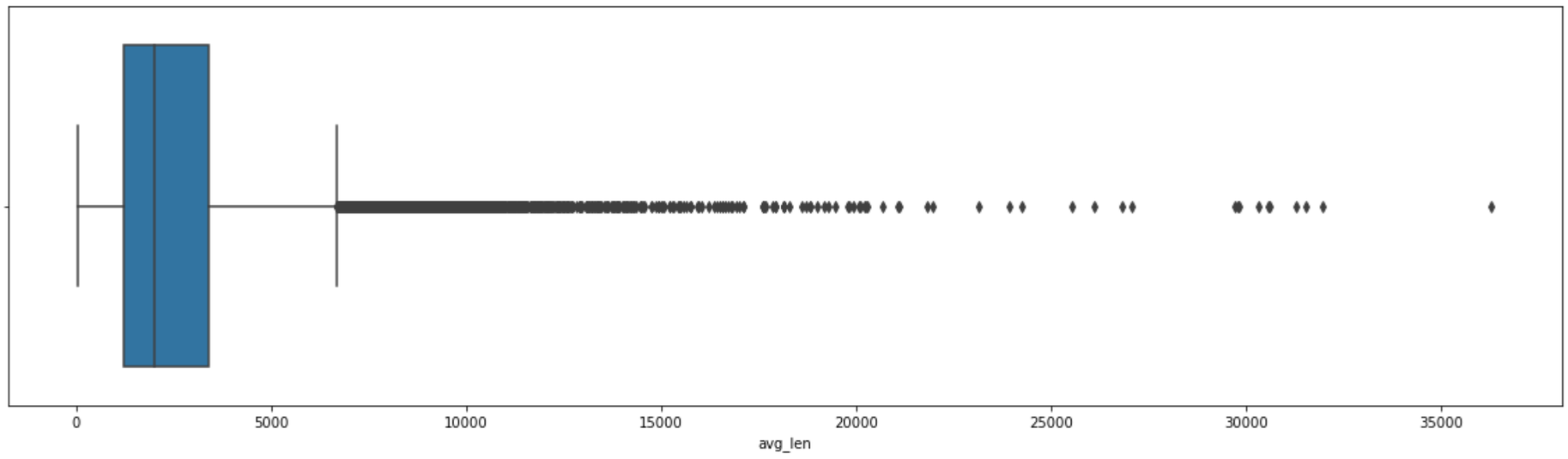
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

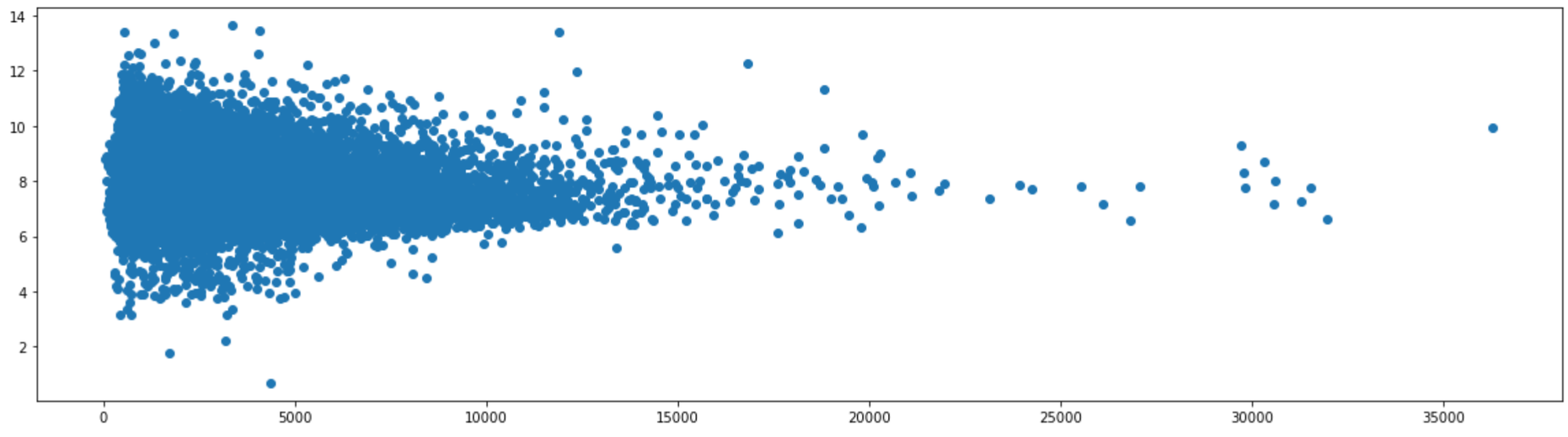
In [362]:
```
1 featureAnalysis('avg_len',dataset)
```

Out[362]:

	count	mean	std	min	25%	50%	75%	max
avg_len	38462.0	2630.67412	2155.376686	66.0	1218.0	1997.0	3413.0	36285.0







polarity

In [363]:
```
1 correlation(dataset,'avg_negative_polarity')
2 correlation(dataset,'avg_positive_polarity')
```

```
                feature  corrMatrix  spearmanCorr  kendallCorr
38  avg_negative_polarity    0.050864      0.026986     0.018212
                feature  corrMatrix  spearmanCorr  kendallCorr
35  avg_positive_polarity    0.069549      0.064372     0.043548
```

In [364]:
```
1 dataset['neg_pos_avg'] = dataset["avg_negative_polarity"] -  dataset["avg_positive_polarity"]
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

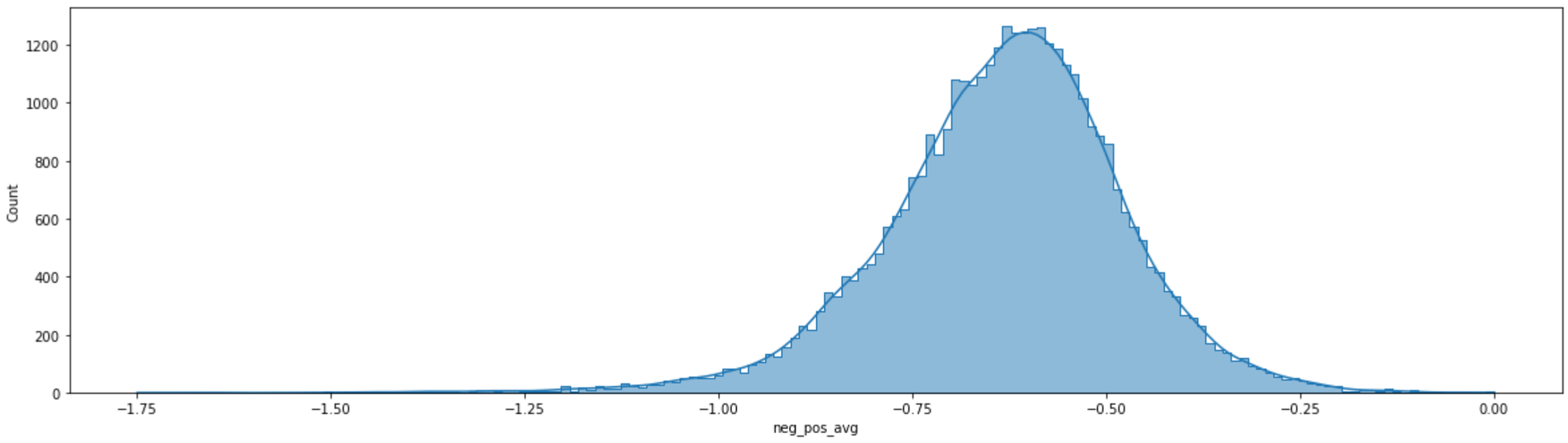
In [365]:
```
1 correlation(dataset,'neg_pos_avg')
```

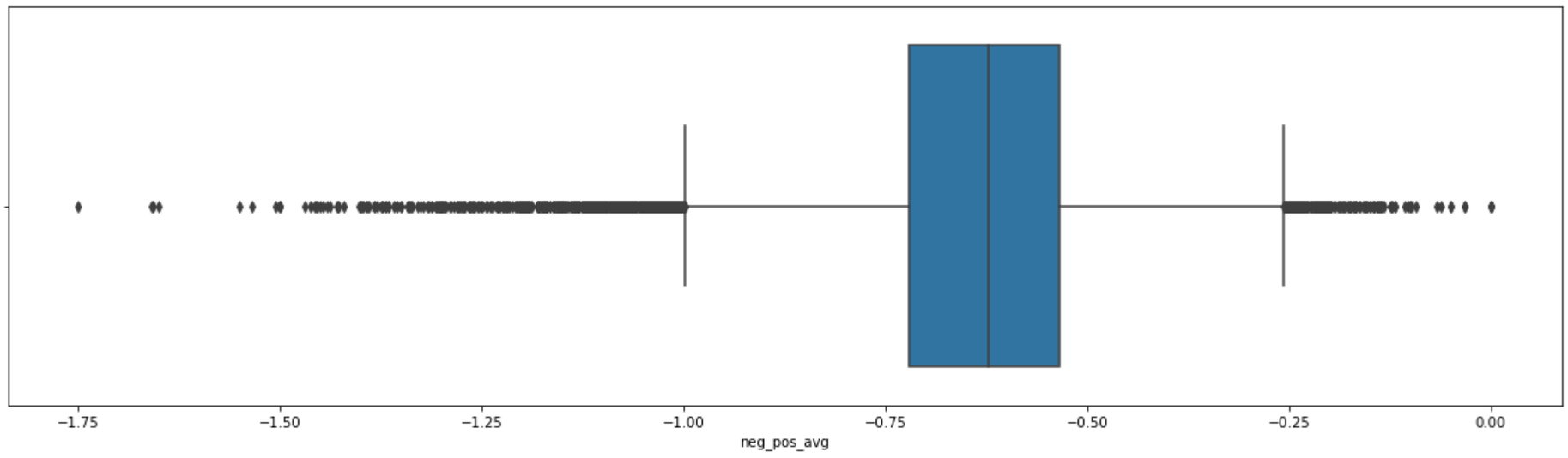
```
        feature  corrMatrix  spearmanCorr  kendallCorr
48  neg_pos_avg    0.078543      0.058643     0.039549
```

In [366]:
```
1 featureAnalysis('neg_pos_avg',dataset)
```

Out[366]:

	count	mean	std	min	25%	50%	75%	max
neg_pos_avg	38462.0	-0.632198	0.154119	-1.75	-0.720455	-0.622532	-0.534724	0.0





In [367]:
```
1 correlation(dataset,'min_negative_polarity')
2 correlation(dataset,'min_positive_polarity')
```

```
                feature  corrMatrix  spearmanCorr  kendallCorr
39  min_negative_polarity    0.036115      0.024486     0.017198
                feature  corrMatrix  spearmanCorr  kendallCorr
36  min_positive_polarity    0.002845      0.039792     0.029546
```

In [368]:
```
1 dataset['neg_pos_min'] = dataset["min_negative_polarity"] -  dataset["min_positive_polarity"]
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

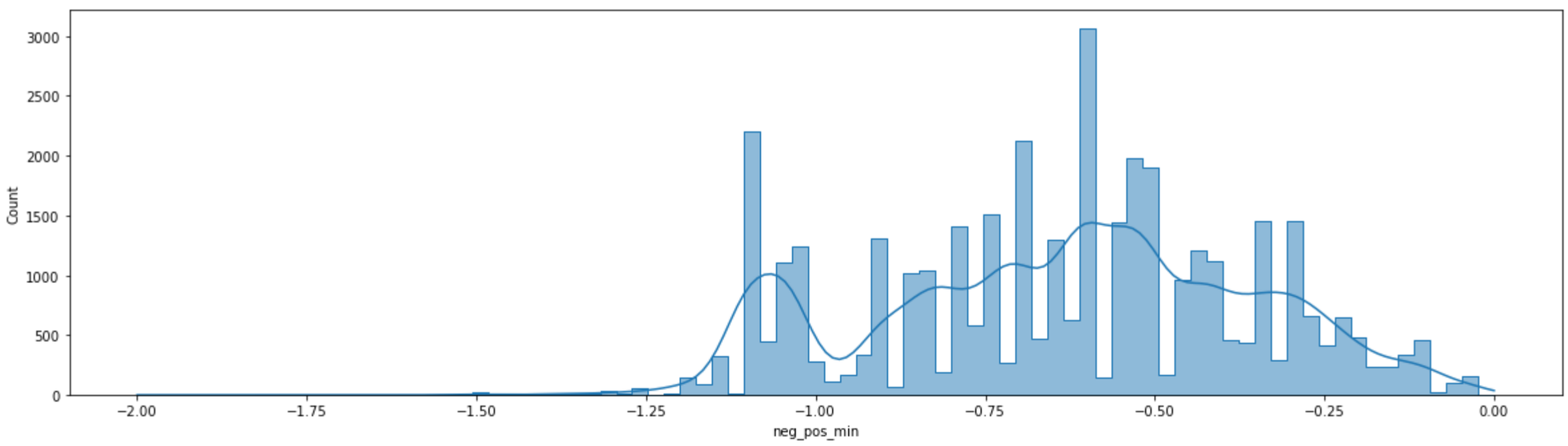
In [369]:
```
1 correlation(dataset,'neg_pos_min')
```

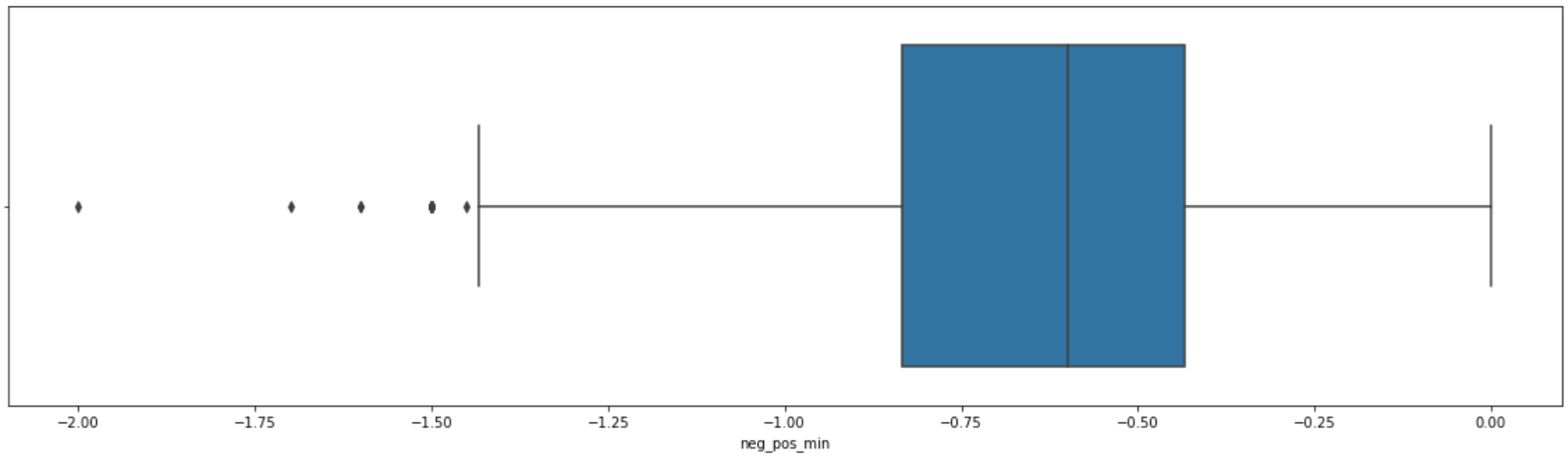
```
       feature  corrMatrix  spearmanCorr  kendallCorr
49  neg_pos_min    0.035884      0.018689     0.012748
```

In [370]:
```
1 featureAnalysis('neg_pos_min',dataset)
```

Out[370]:

	count	mean	std	min	25%	50%	75%	max
neg_pos_min	38462.0	-0.636363	0.275918	-2.0	-0.833333	-0.6	-0.433333	0.0





In [371]:
```
1 correlation(dataset,'max_negative_polarity')
2 correlation(dataset,'max_positive_polarity')
```

```
                feature  corrMatrix  spearmanCorr  kendallCorr
40  max_negative_polarity     0.02186      0.010065     0.007249
                feature  corrMatrix  spearmanCorr  kendallCorr
37  max_positive_polarity    0.064723      0.069101     0.050827
```

In [372]:
```
1 dataset['neg_pos_max'] = dataset["max_negative_polarity"] -  dataset["max_positive_polarity"]
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

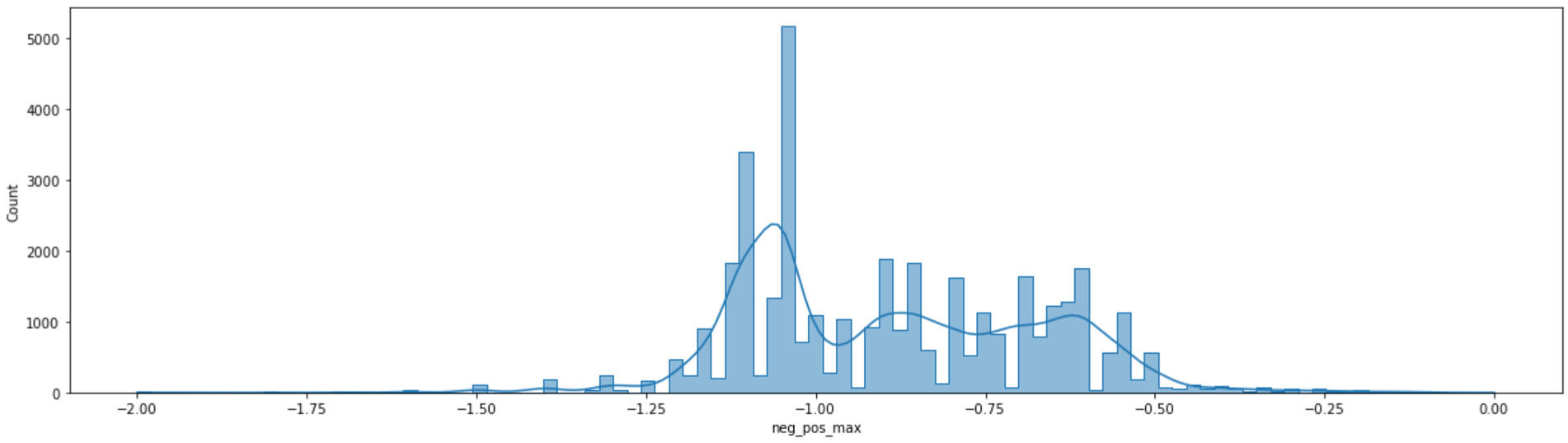
In [373]:
```
1 correlation(dataset,'neg_pos_max')
```

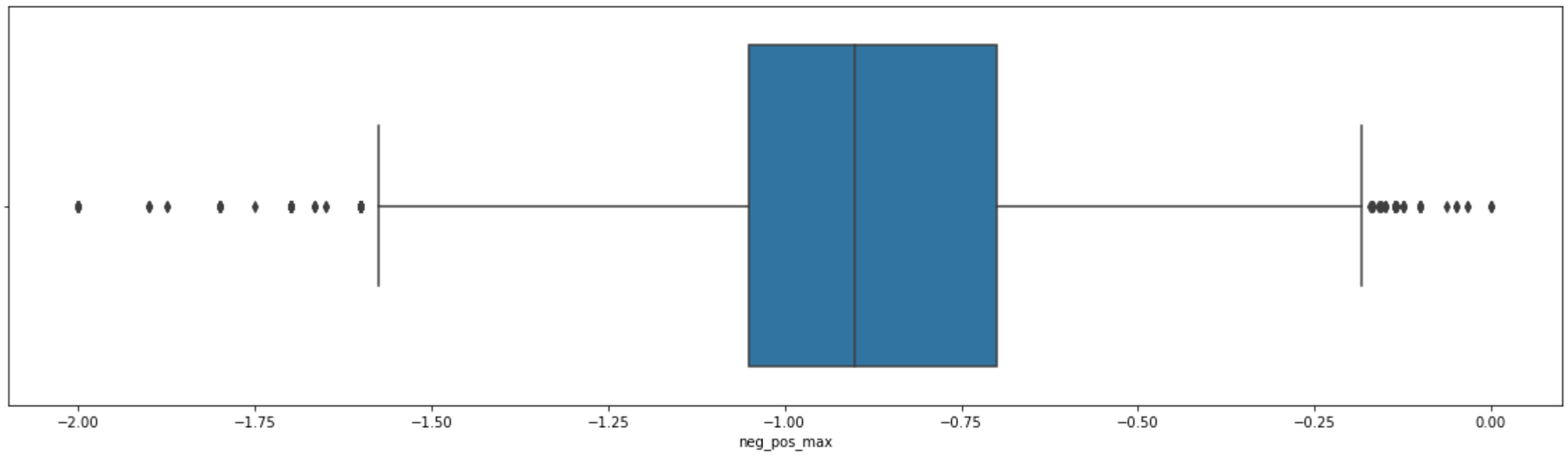
```
       feature  corrMatrix  spearmanCorr  kendallCorr
50  neg_pos_max    0.071311      0.070209     0.048242
```

In [374]:
```
1 featureAnalysis('neg_pos_max',dataset)
```

Out[374]:

	count	mean	std	min	25%	50%	75%	max
neg_pos_max	38462.0	-0.890787	0.221939	-2.0	-1.05	-0.9	-0.7	0.0





Delete last two items

In [375]:
```
1 correlation(dataset,'all')
```

	feature	corrMatrix	spearmanCorr	kendallCorr
1	n_tokens_title	0.022806	0.042942	0.030722
2	n_tokens_content	0.033075	0.018588	0.012758
3	n_unique_tokens	0.022887	0.039634	0.027074
4	num_hrefs	0.114832	0.104030	0.072039
5	num_self_hrefs	0.035281	0.046446	0.033177
6	num_imgs	0.099305	0.089835	0.067289
7	num_videos	0.057621	0.047234	0.036996
8	average_token_length	0.040346	0.052766	0.035447
9	num_keywords	0.066329	0.072598	0.052207
10	weekdays	0.084427	0.105127	0.076304
11	data_channel	0.132609	0.109931	0.080280
12	kw_min_min	0.024657	0.006308	0.004514
13	kw_max_min	0.032038	0.092264	0.062419
14	kw_avg_min	0.039456	0.095153	0.064177
15	kw_min_max	0.006839	0.055825	0.040500
16	kw_max_max	0.000768	0.032475	0.025527
17	kw_avg_max	0.053264	0.034101	0.022747
18	kw_min_avg	0.107444	0.102328	0.074831
19	kw_max_avg	0.107587	0.224851	0.153878
20	kw_avg_avg	0.221412	0.258610	0.177038
21	self_reference_min_shares	0.076187	0.199026	0.138177
22	self_reference_max_shares	0.080674	0.185450	0.128126
23	self_reference_avg_sharess	0.090760	0.210281	0.145569
24	is_weekend	0.116268	0.153851	0.126639
25	LDA_00	0.033392	0.022962	0.015897
26	LDA_01	0.051764	0.068867	0.046364
27	LDA_02	0.167744	0.160707	0.108240
28	LDA_03	0.124014	0.065222	0.043525
29	LDA_04	0.048222	0.055119	0.037982
30	global_subjectivity	0.134202	0.129127	0.087165
31	global_sentiment_polarity	0.058896	0.090345	0.060856
32	global_rate_positive_words	0.060838	0.084283	0.056668
33	global_rate_negative_words	0.003404	0.015659	0.010427
34	rate_positive_words	0.040164	0.067778	0.045547
35	avg_positive_polarity	0.069549	0.064372	0.043548
36	min_positive_polarity	0.002845	0.039792	0.029546
37	max_positive_polarity	0.064723	0.069101	0.050827
38	avg_negative_polarity	0.050864	0.026986	0.018212
39	min_negative_polarity	0.036115	0.024486	0.017198
40	max_negative_polarity	0.021860	0.010065	0.007249
41	title_subjectivity	0.052586	0.040313	0.029209
42	title_sentiment_polarity	0.050184	0.057627	0.042270
43	abs_title_subjectivity	0.000036	0.000913	0.000705
44	abs_title_sentiment_polarity	0.058808	0.036317	0.026568
45	shares	1.000000	1.000000	1.000000
46	media	0.134062	0.121660	0.088980
47	avg_len	0.028812	0.014834	0.010187
48	neg_pos_avg	0.078543	0.058643	0.039549
49	neg_pos_min	0.035884	0.018689	0.012748
50	neg_pos_max	0.071311	0.070209	0.048242

In [376]:
```
1 outlierDetection(dataset_RAW)
```

Out[376]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('abs_title_subjectivity', 0),
 ('num_keywords', 50),
 ('n_tokens_title', 155),
 ('n_unique_tokens', 490),
 ('rate_positive_words', 519),
 ('global_rate_positive_words', 520),
 ('average_token_length', 552),
 ('global_subjectivity', 864),
 ('kw_avg_max', 879),
 ('avg_negative_polarity', 879),
 ('global_sentiment_polarity', 944),
 ('avg_positive_polarity', 1042),
 ('global_rate_negative_words', 1358),
 ('abs_title_sentiment_polarity', 1589),
 ('shares', 1599),
 ('kw_avg_avg', 1642),
 ('n_tokens_content', 1875),
 ('kw_avg_min', 2009),
 ('num_self_hrefs', 2090),
 ('kw_max_avg', 2352),
 ('max_negative_polarity', 2454),
 ('num_hrefs', 2636),
 ('num_videos', 2939),
 ('LDA_03', 3029),
 ('min_positive_polarity', 3147),
 ('LDA_02', 3441),
 ('kw_max_min', 3566),
 ('self_reference_max_shares', 4094),
 ('self_reference_avg_sharess', 4175),
 ('kw_min_min', 4638),
 ('kw_min_max', 4827),
 ('self_reference_min_shares', 4884),
 ('is_weekend', 5026),
 ('LDA_00', 5071),
 ('LDA_01', 5682),
 ('num_imgs', 7466),
 ('title_sentiment_polarity', 7918),
 ('kw_max_max', 9365)]
```

In [377]:
```
1 # kw_max_max                0.000768      0.032475     0.025527  outlier : 9365
2 # abs_title_subjectivity    0.000036      0.000913     0.000705  outlier : 1589
```

In [378]:
```
1 dataset.drop('kw_max_max', inplace=True, axis=1)
2 dataset.drop('abs_title_subjectivity', inplace=True, axis=1)
```

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

In [379]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.6542459913255024
MSE : 0.7712790702335444
RMSE : 0.8782249542307167
r2_score : 0.11219872817225007  11.0

Ridge Model.....................................

The train score for ridge model is 0.12245159308102938
The test score for ridge model is 0.11208314576499856
MAE : 0.6541825940496352
MSE : 0.7713794826731625
r2_score : 0.11208314576499856  11.0

Lasso Model.....................................

The train score for ls model is 0.08339856811972257
The test score for ls model is 0.07684970875771802
MAE : 0.6727242559460294
MSE : 0.8019885991482513
r2_score : 0.07684970875771802  8.0
```

Fix outliers

In [380]:
```
dicto = {}
def outlierDetection(dataset):
    num_cols = dataset.select_dtypes(['int64','float64']).columns
    for column in num_cols:
        q1 = dataset[column].quantile(0.25)    # First Quartile
        q3 = dataset[column].quantile(0.75)    # Third Quartile
        IQR = q3 - q1                            # Inter Quartile Range
        llimit = q1 - 1.5*IQR                       # Lower Limit
        ulimit = q3 + 1.5*IQR                        # Upper Limit
        outliers = dataset[(dataset[column] < llimit) | (dataset[column] > ulimit)]
        dicto[column] = len(outliers)
    sorted_dicto = sorted(dicto.items(), key=lambda x:x[1])
    return (sorted_dicto)
```

In [381]:
```
datset_Outlier = dataset.copy()
```

In [382]:
```
outlierDetection(dataset)
```

Out[382]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('neg_pos_min', 24),
 ('media', 28),
 ('num_keywords', 50),
 ('n_tokens_title', 155),
 ('neg_pos_max', 167),
 ('n_unique_tokens', 489),
 ('rate_positive_words', 518),
 ('global_rate_positive_words', 520),
 ('average_token_length', 552),
 ('global_subjectivity', 863),
 ('kw_avg_max', 879),
 ('avg_negative_polarity', 879),
 ('global_sentiment_polarity', 944),
 ('neg_pos_avg', 960),
 ('avg_positive_polarity', 1041),
 ('global_rate_negative_words', 1358),
 ('abs_title_sentiment_polarity', 1589),
 ('shares', 1599),
 ('kw_avg_avg', 1641),
 ('avg_len', 1846),
 ('n_tokens_content', 1874),
 ('num_videos', 1903),
 ('kw_avg_min', 2009),
 ('num_self_hrefs', 2089),
 ('kw_max_avg', 2352),
 ('max_negative_polarity', 2454),
 ('num_hrefs', 2636),
 ('LDA_03', 3028),
 ('min_positive_polarity', 3147),
 ('LDA_02', 3441),
 ('kw_max_min', 3566),
 ('self_reference_max_shares', 4094),
 ('self_reference_avg_sharess', 4175),
 ('num_imgs', 4407),
 ('kw_min_min', 4638),
 ('kw_min_max', 4826),
 ('self_reference_min_shares', 4884),
 ('is_weekend', 5026),
 ('LDA_00', 5071),
 ('LDA_01', 5682),
 ('title_sentiment_polarity', 7918)]
```

In [384]:
```
dataset.columns
```

Out[384]:
```
Index(['n_tokens_title', 'n_tokens_content', 'n_unique_tokens', 'num_hrefs',
       'num_self_hrefs', 'num_imgs', 'num_videos', 'average_token_length',
       'num_keywords', 'weekdays', 'data_channel', 'kw_min_min', 'kw_max_min',
       'kw_avg_min', 'kw_min_max', 'kw_avg_max', 'kw_min_avg', 'kw_max_avg',
       'kw_avg_avg', 'self_reference_min_shares', 'self_reference_max_shares',
       'self_reference_avg_sharess', 'is_weekend', 'LDA_00', 'LDA_01',
       'LDA_02', 'LDA_03', 'LDA_04', 'global_subjectivity',
       'global_sentiment_polarity', 'global_rate_positive_words',
       'global_rate_negative_words', 'rate_positive_words',
       'avg_positive_polarity', 'min_positive_polarity',
       'max_positive_polarity', 'avg_negative_polarity',
       'min_negative_polarity', 'max_negative_polarity', 'title_subjectivity',
       'title_sentiment_polarity', 'abs_title_sentiment_polarity', 'shares',
       'media', 'avg_len', 'neg_pos_avg', 'neg_pos_min', 'neg_pos_max'],
      dtype='object')
```

Links outliers

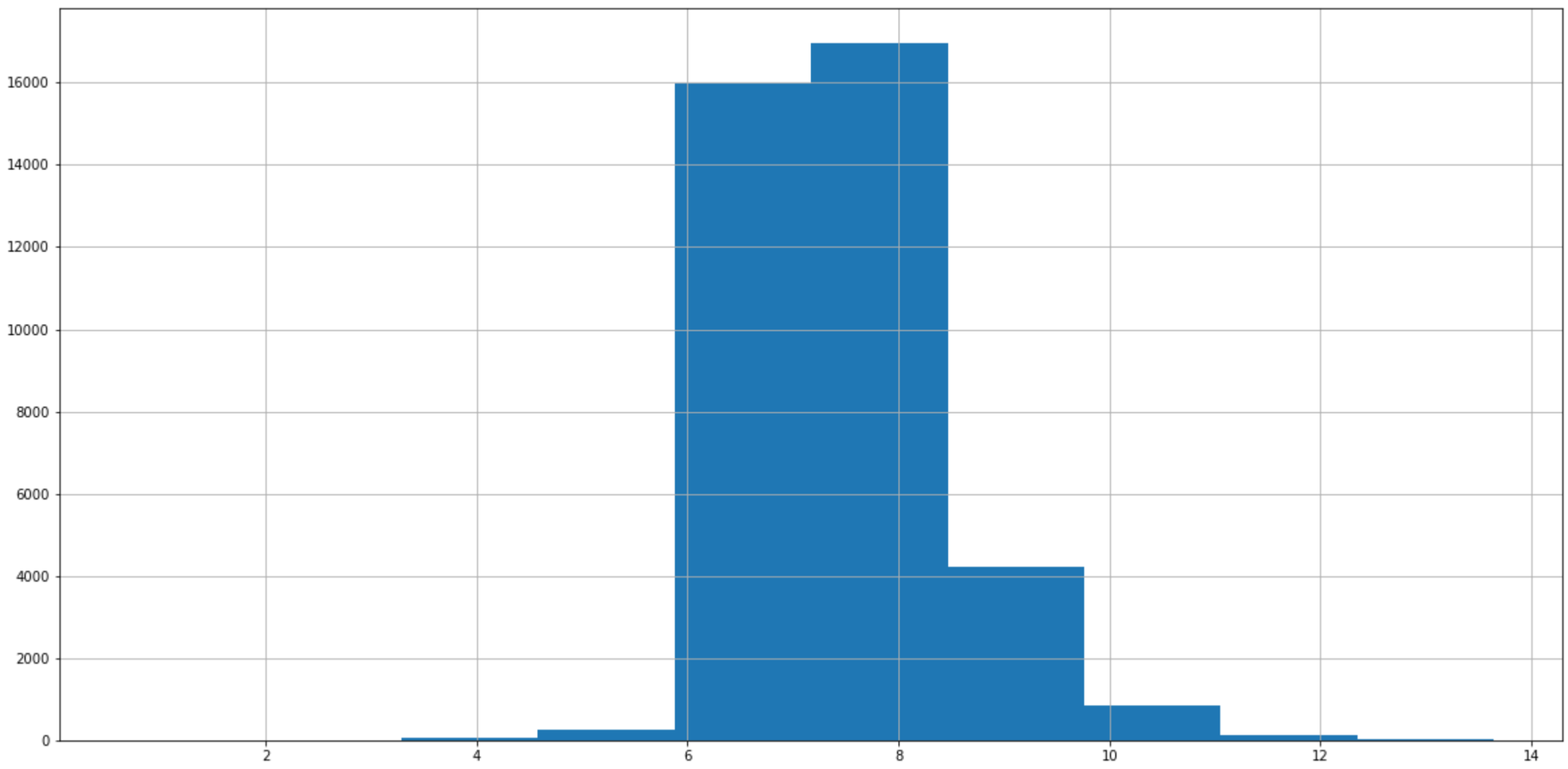
In [385]:
```
1 len(dataset[dataset['num_self_hrefs'] == 0])
```

Out[385]: 4169

In [386]:
```
1 selfHref_0 = dataset[dataset['num_self_hrefs'] == 0]
```

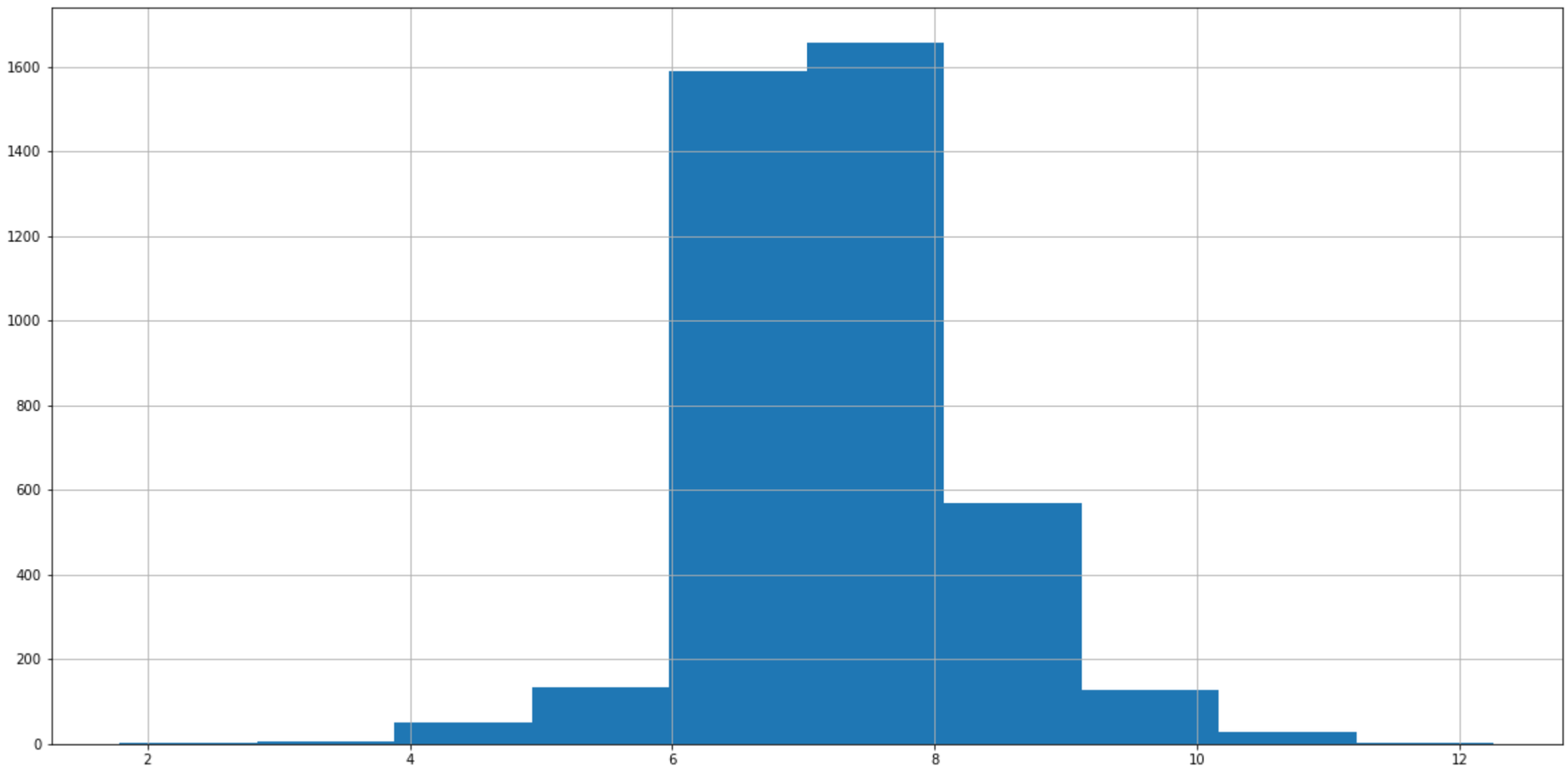
In [387]:
```
1 dataset['shares'].hist(figsize=(20,10))
```

Out[387]: <AxesSubplot:>



In [388]:
```
1 selfHref_0['shares'].hist(figsize=(20,10))
```

Out[388]: <AxesSubplot:>



In [389]:
```
1 len(selfHref_0[selfHref_0['self_reference_min_shares'] == 0])
```

Out[389]: 4169

In [390]:
```
1 len(selfHref_0[selfHref_0['self_reference_min_shares'] == 0])
```

Out[390]: 4169

In [391]:
```
1 len(dataset[dataset['self_reference_min_shares'] == 0])
```

Out[391]: 5993

In [392]:
```
1 len(dataset[dataset['self_reference_max_shares'] == 0])
```

Out[392]: 5993

In [393]:
```
1 len(dataset[dataset['self_reference_avg_sharess'] == 0])
```

Out[393]: 5993

In [394]:
```
1 dataset = dataset[dataset['num_self_hrefs'] != 0]
```

In [395]:
```
1 dataset
```

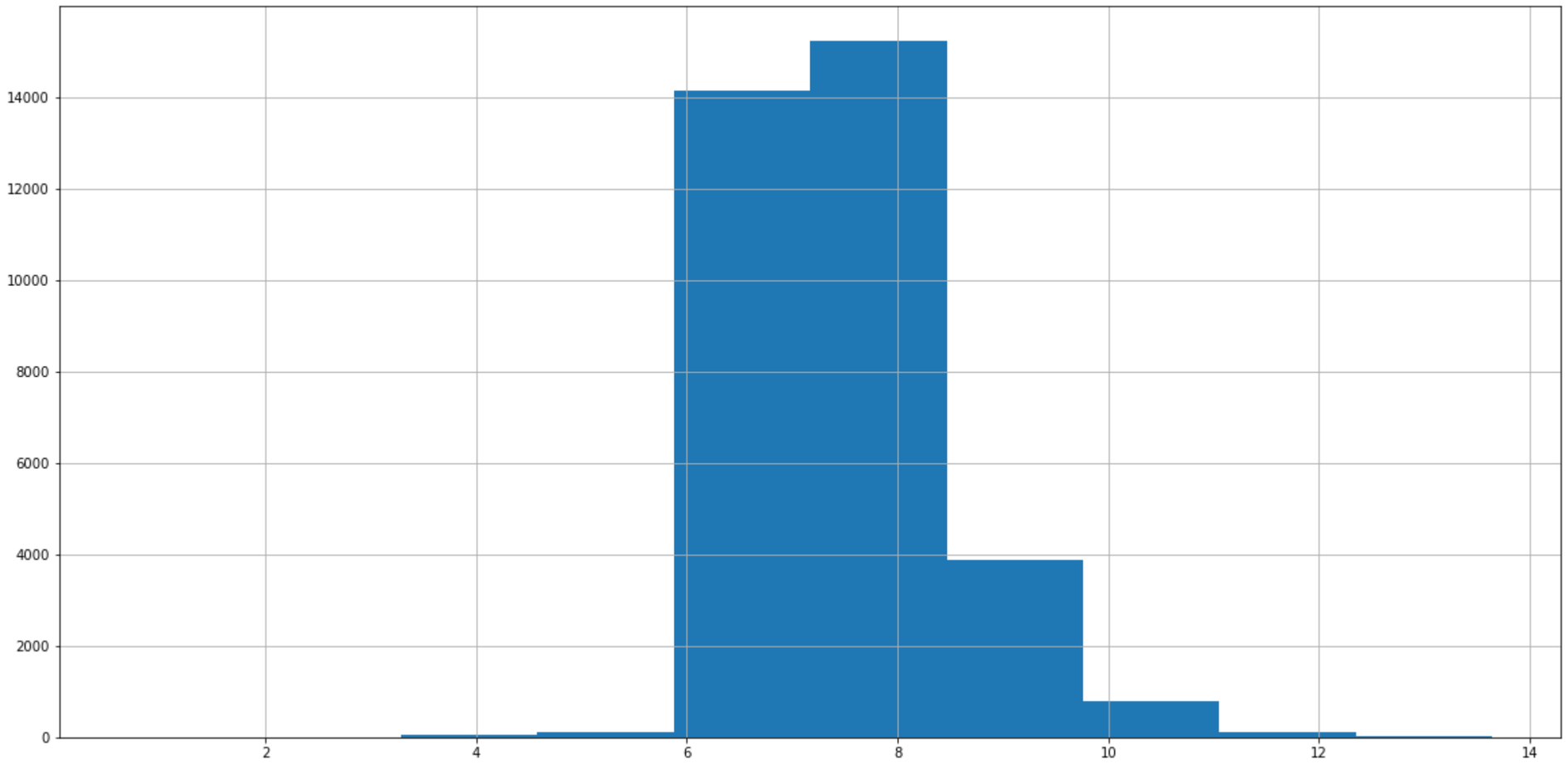
Out[395]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
0	12.0	219.0	0.663594	4.0	2.0	1.000000	0.000000	4.680365	5.0	1	2	0.0	0.0	0.000	
1	9.0	255.0	0.604743	3.0	1.0	1.000000	0.000000	4.913725	4.0	1	3	0.0	0.0	0.000	
2	9.0	211.0	0.575130	3.0	1.0	1.000000	0.000000	4.393365	6.0	1	3	0.0	0.0	0.000	
4	13.0	1072.0	0.415646	19.0	19.0	4.472136	0.000000	4.682836	7.0	1	5	0.0	0.0	0.000	
5	10.0	370.0	0.559889	2.0	2.0	0.000000	0.000000	4.359459	9.0	1	5	0.0	0.0	0.000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
39639	11.0	346.0	0.529052	9.0	7.0	1.000000	1.000000	4.523121	8.0	3	5	-1.0	671.0	173.125	
39640	12.0	328.0	0.696296	9.0	7.0	1.732051	6.928203	4.405488	7.0	3	4	-1.0	616.0	184.000	
39641	10.0	442.0	0.516355	24.0	1.0	3.464102	1.000000	5.076923	8.0	3	0	-1.0	691.0	168.250	
39642	6.0	682.0	0.539493	10.0	1.0	1.000000	0.000000	4.975073	5.0	3	6	-1.0	0.0	-1.000	
39643	10.0	157.0	0.701987	1.0	1.0	0.000000	1.414214	4.471338	4.0	3	2	-1.0	97.0	23.500	

34293 rows × 48 columns

In [396]:
```
1 dataset['shares'].hist(figsize=(20,10))
```

Out[396]: <AxesSubplot:>



In [397]:
```
1 outlierDetection(dataset)
```

Out[397]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_03', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('media', 0),
 ('neg_pos_min', 24),
 ('num_keywords', 29),
 ('n_tokens_title', 147),
 ('neg_pos_max', 151),
 ('n_unique_tokens', 418),
 ('rate_positive_words', 467),
 ('global_rate_positive_words', 475),
 ('average_token_length', 525),
 ('kw_avg_max', 758),
 ('global_subjectivity', 765),
 ('avg_negative_polarity', 785),
 ('neg_pos_avg', 806),
 ('global_sentiment_polarity', 818),
 ('avg_positive_polarity', 884),
 ('global_rate_negative_words', 1262),
 ('shares', 1300),
 ('kw_avg_avg', 1399),
 ('abs_title_sentiment_polarity', 1417),
 ('num_imgs', 1552),
 ('num_self_hrefs', 1637),
 ('avg_len', 1732),
 ('n_tokens_content', 1770),
 ('kw_avg_min', 1773),
 ('num_videos', 1878),
 ('kw_max_avg', 2116),
 ('max_negative_polarity', 2279),
 ('num_hrefs', 2555),
 ('min_positive_polarity', 2943),
 ('kw_max_min', 2985),
 ('LDA_02', 3517),
 ('self_reference_max_shares', 3545),
 ('self_reference_avg_sharess', 3648),
 ('kw_min_min', 3851),
 ('kw_min_max', 4141),
 ('self_reference_min_shares', 4287),
 ('LDA_00', 4406),
 ('is_weekend', 4575),
 ('LDA_01', 4835),
 ('title_sentiment_polarity', 6591)]
```

kw_max_min

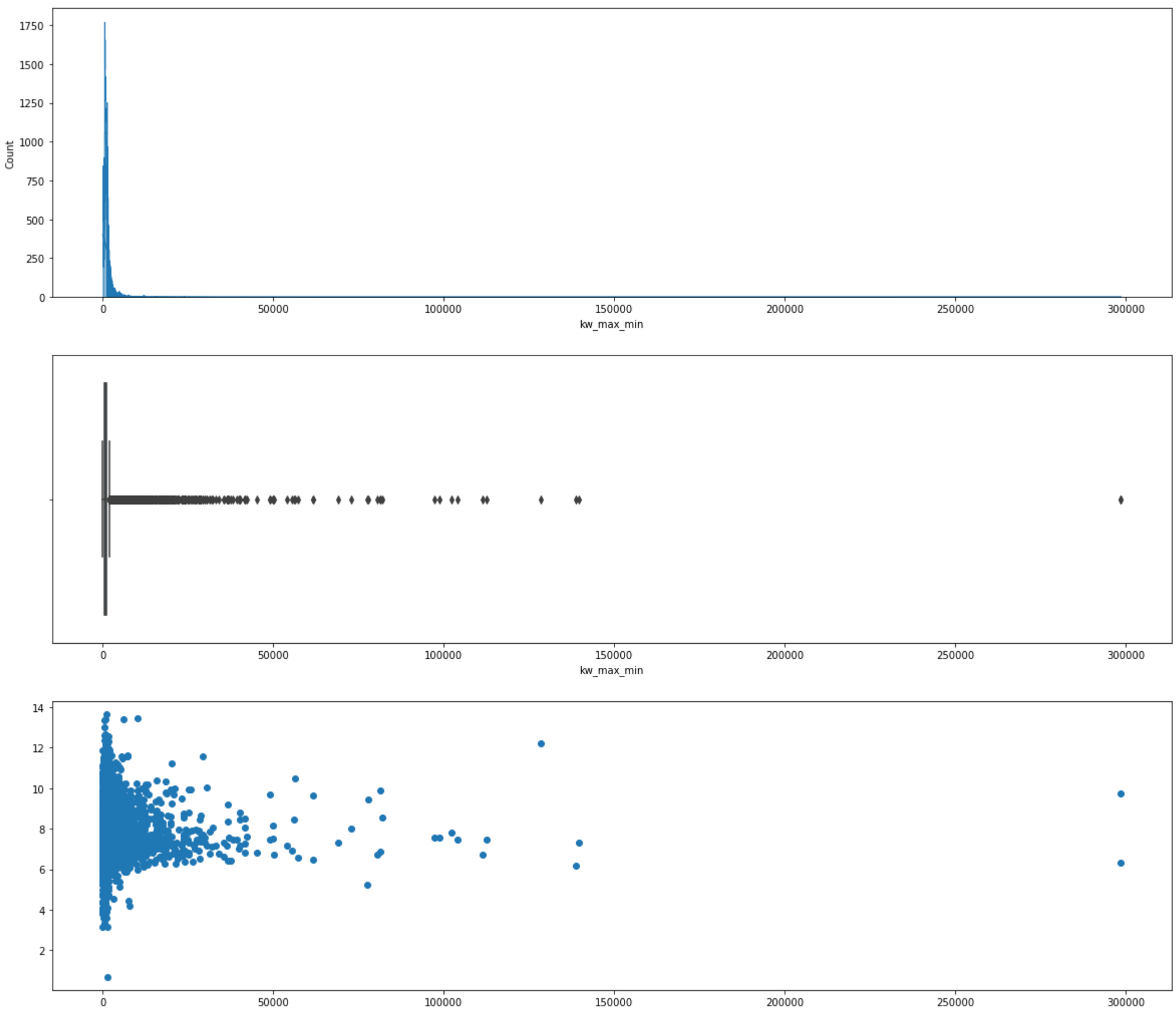
In [398]:
```
1 len(dataset[dataset['kw_max_min'] == 0])
```

Out[398]: 715

In [399]:
```
1 featureAnalysis('kw_max_min',dataset)
```

Out[399]:

	count	mean	std	min	25%	50%	75%	max
kw_max_min	34293.0	1129.050092	3890.560806	0.0	444.0	653.0	1000.0	298400.0

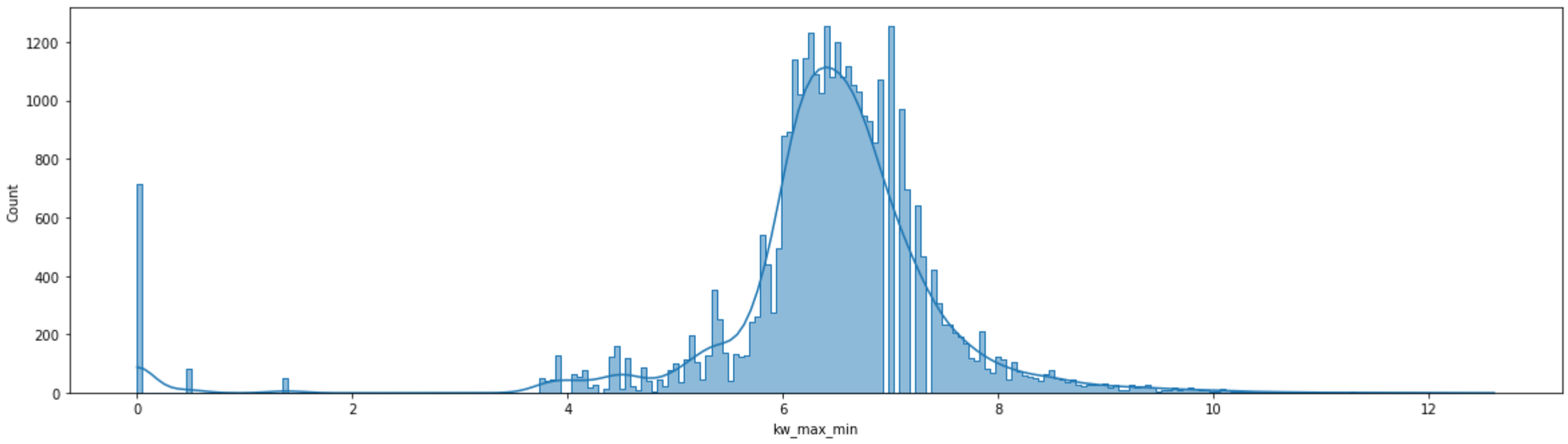


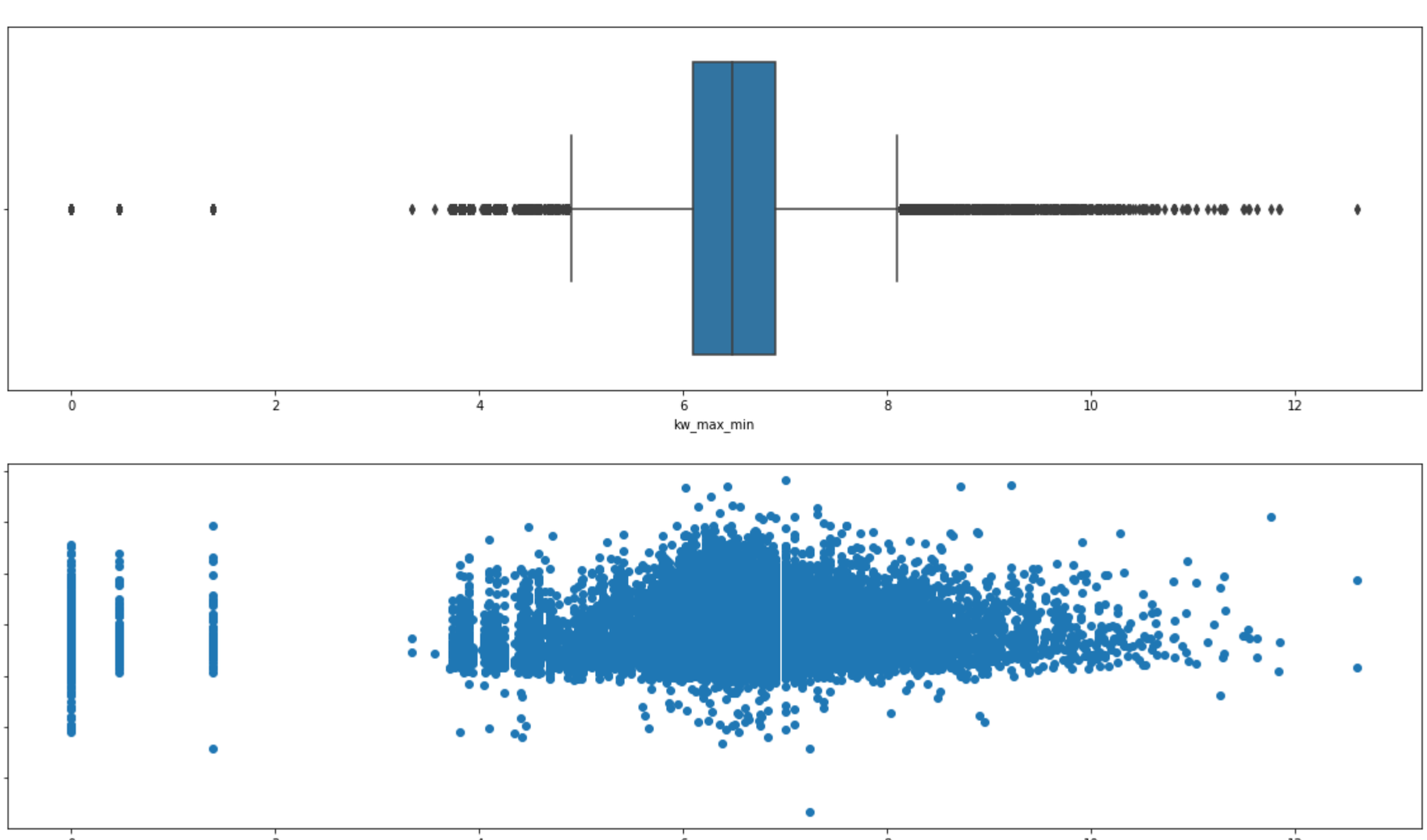
In [400]:
```
1 dataset['kw_max_min']= np.log(dataset.kw_max_min,where=dataset.kw_max_min>0)
```

In [401]:
```
1 featureAnalysis('kw_max_min',dataset)
```

Out[401]:

	count	mean	std	min	25%	50%	75%	max
kw_max_min	34293.0	6.37779	1.303592	0.0	6.095825	6.481577	6.907755	12.60619





kw_max_avg

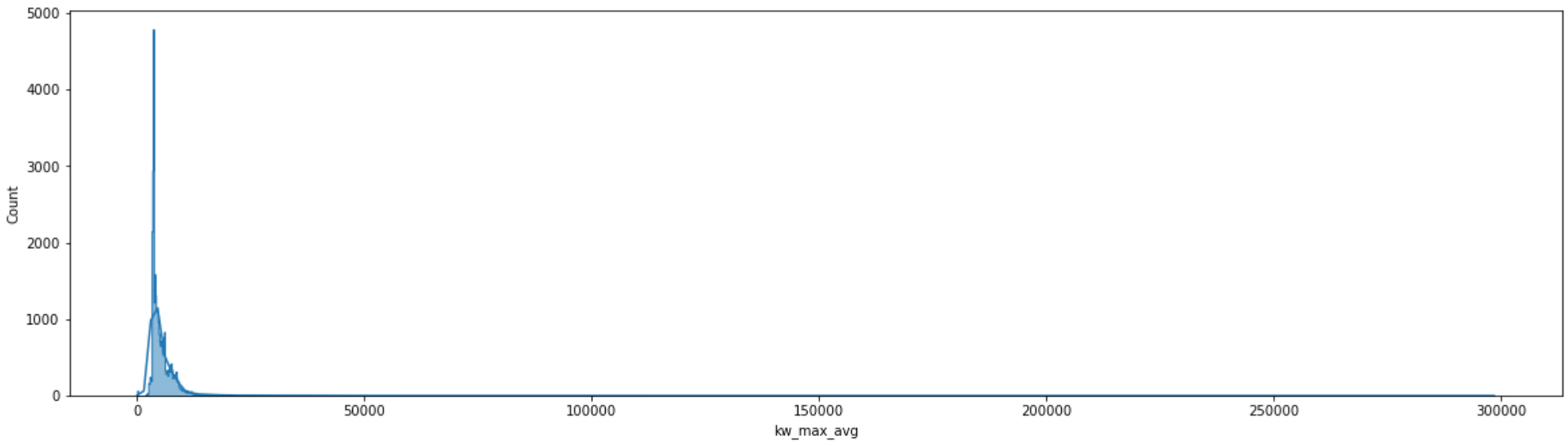
In [402]:
```
1 len(dataset[dataset['kw_max_avg'] == 0])
```

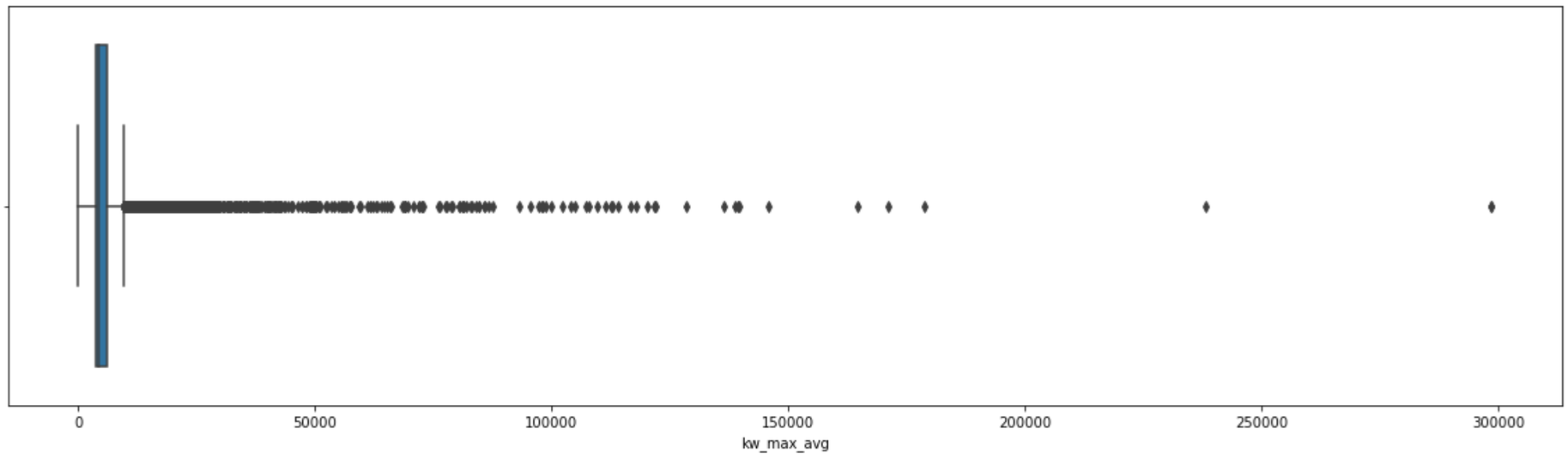
Out[402]: 67

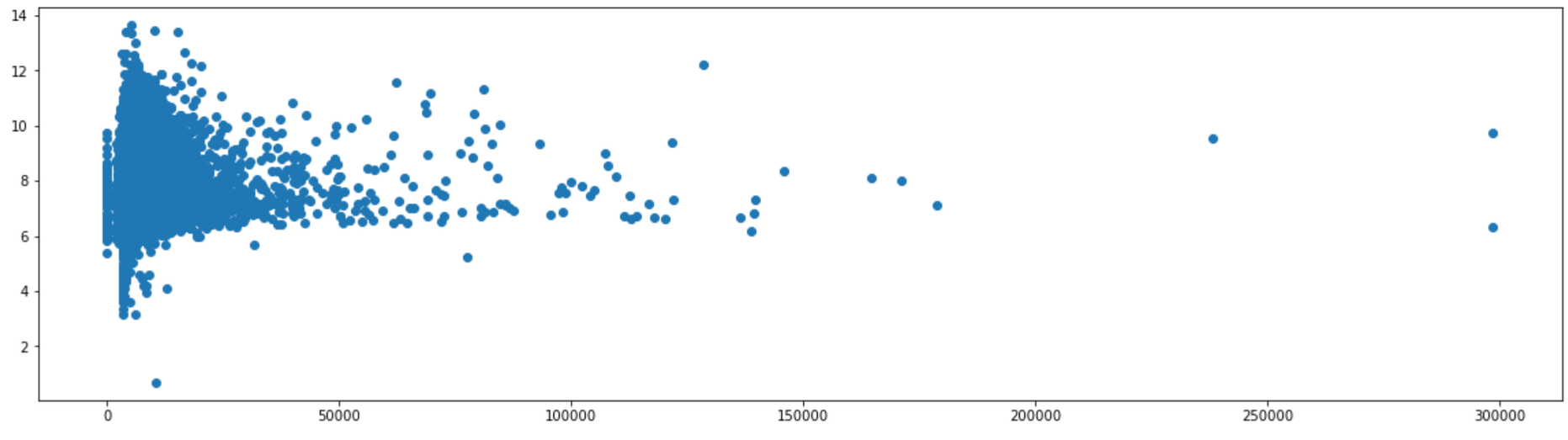
In [403]:
```
1 featureAnalysis('kw_max_avg',dataset)
```

Out[403]:

	count	mean	std	min	25%	50%	75%	max
kw_max_avg	34293.0	5677.453168	6250.267969	0.0	3566.297886	4349.06538	6020.524129	298400.0





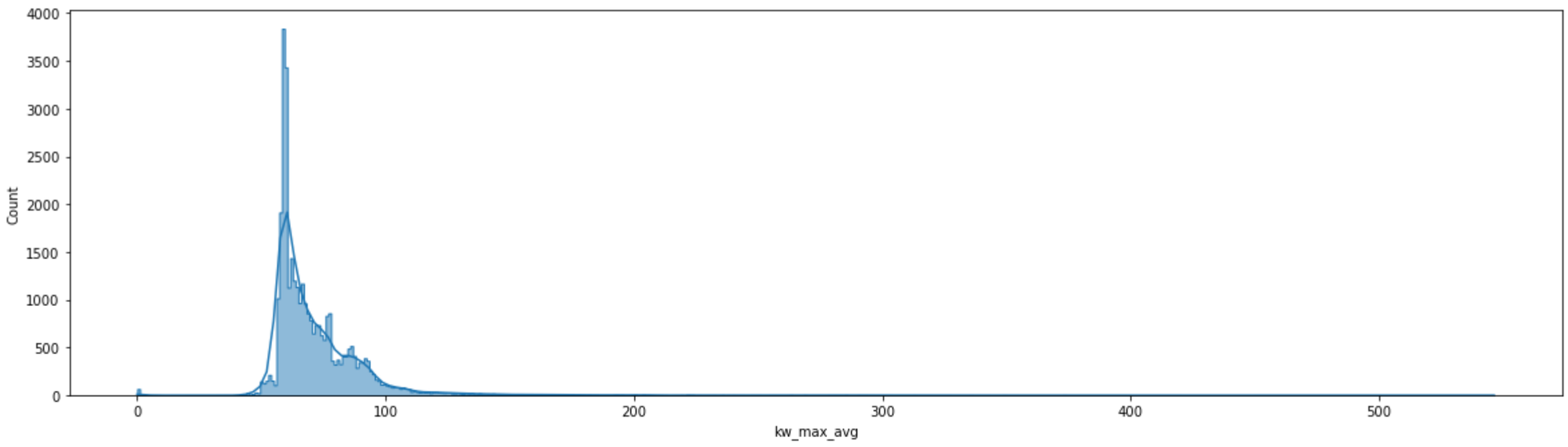


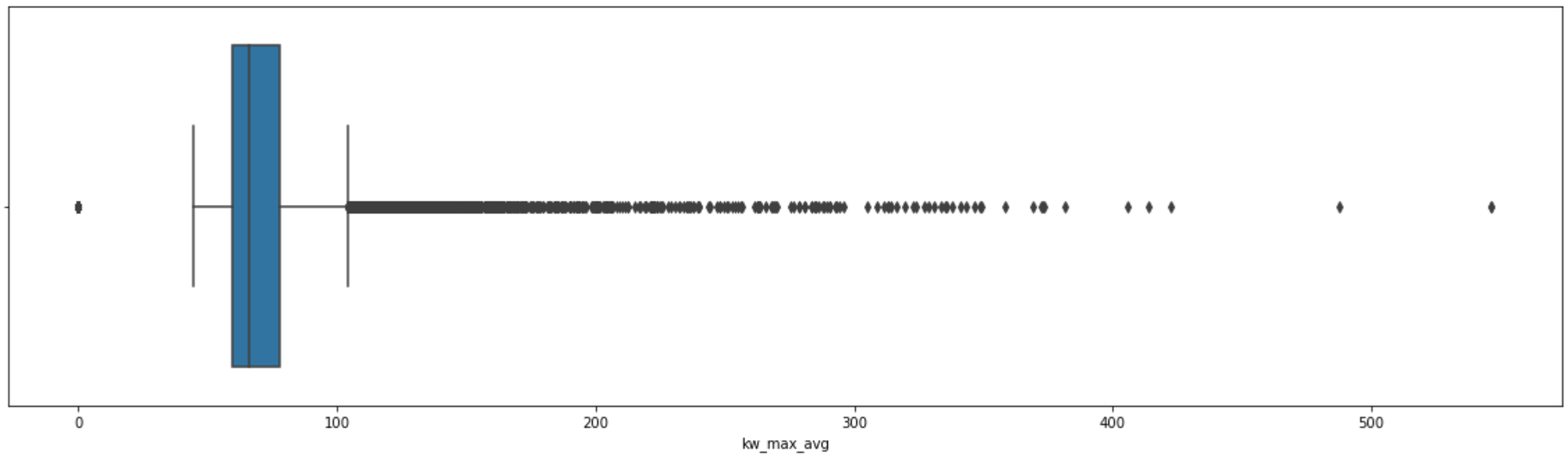
In [404]:
```
1 dataset['kw_max_avg']= np.sqrt(dataset.kw_max_avg)
```

In [405]:
```
1 featureAnalysis('kw_max_avg',dataset)
```

Out[405]:

	count	mean	std	min	25%	50%	75%	max
kw_max_avg	34293.0	72.060212	22.01802	0.0	59.718489	65.947444	77.592037	546.260011





kw_avg_avg

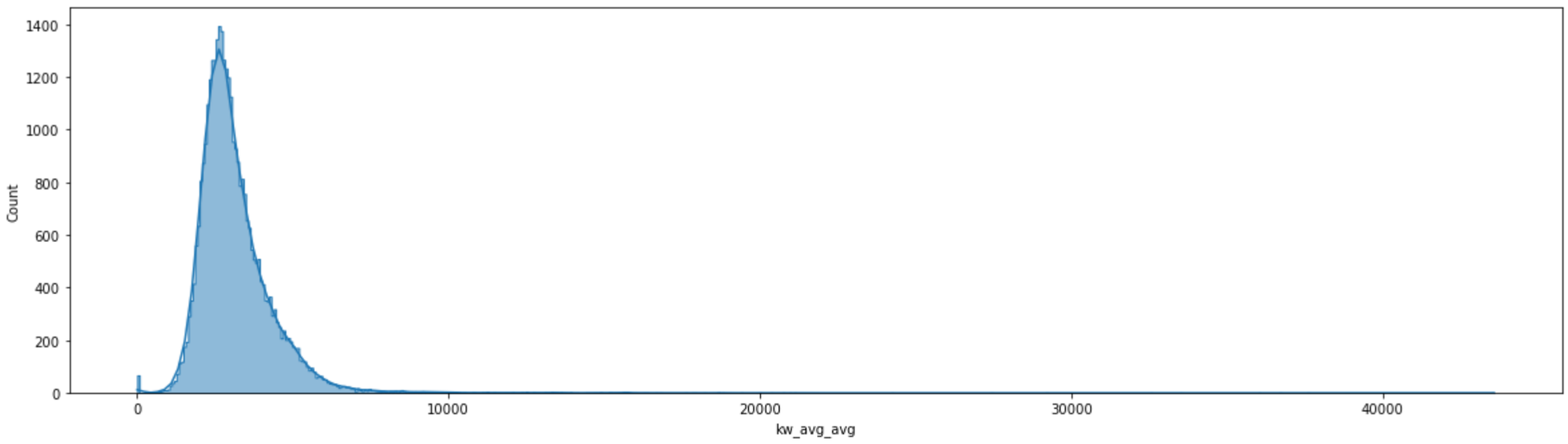
In [406]:
```
1 len(dataset[dataset['kw_avg_avg'] == 0])
```

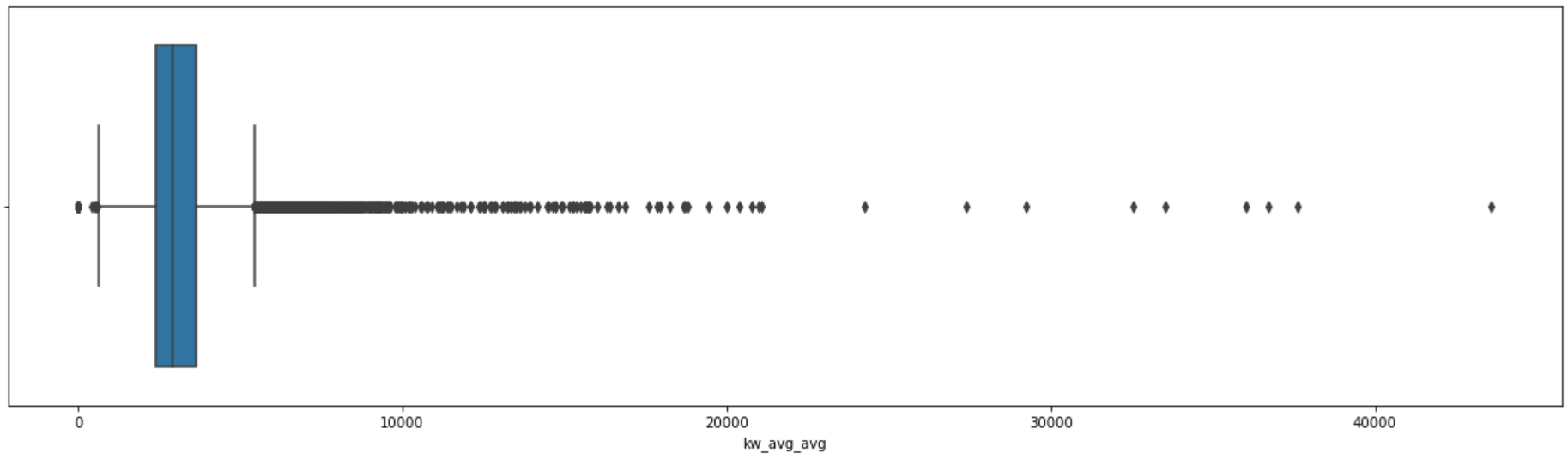
Out[406]: 67

In [407]:
```
1 featureAnalysis('kw_avg_avg',dataset)
```

Out[407]:

	count	mean	std	min	25%	50%	75%	max
kw_avg_avg	34293.0	3151.884343	1323.263572	0.0	2404.733413	2890.625105	3611.559266	43567.659946



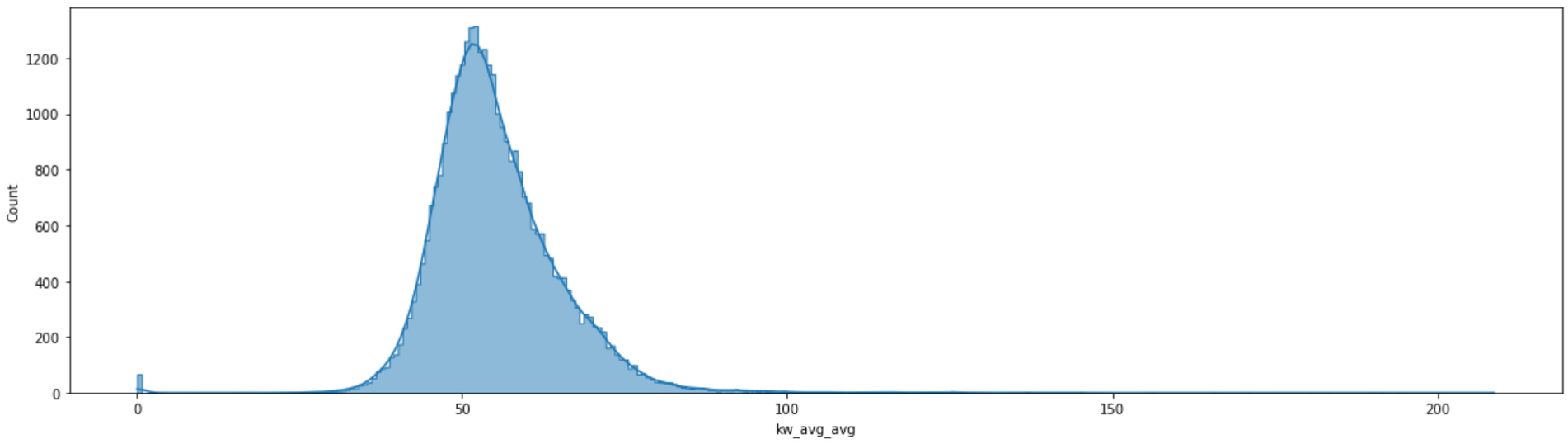


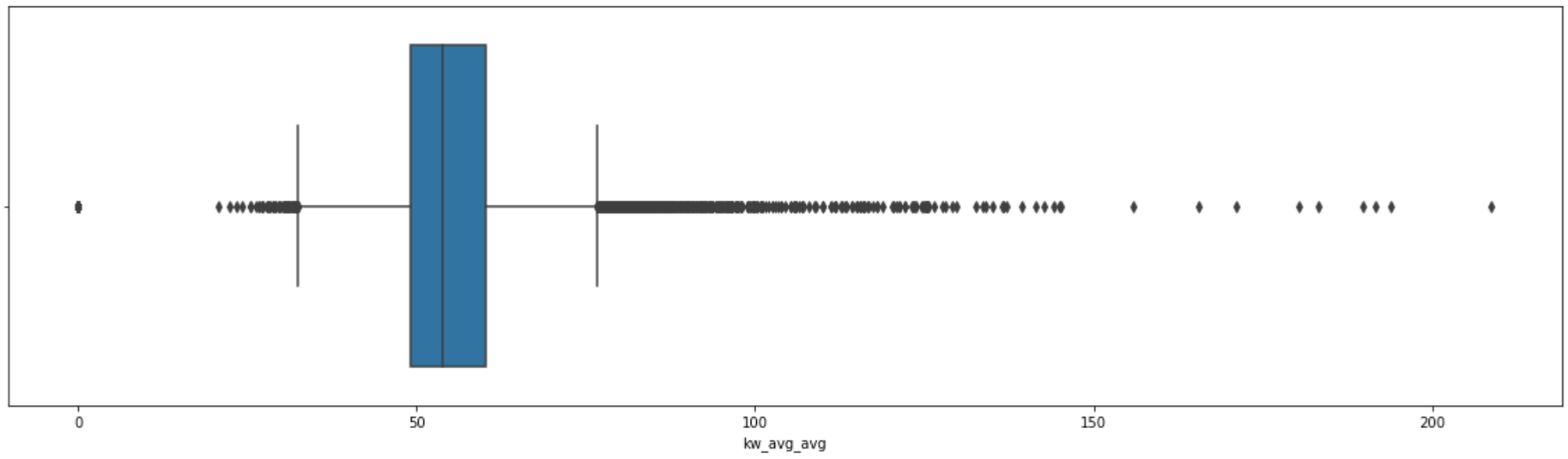
In [408]:
```
1 dataset['kw_avg_avg']= np.sqrt(dataset.kw_avg_avg)
```

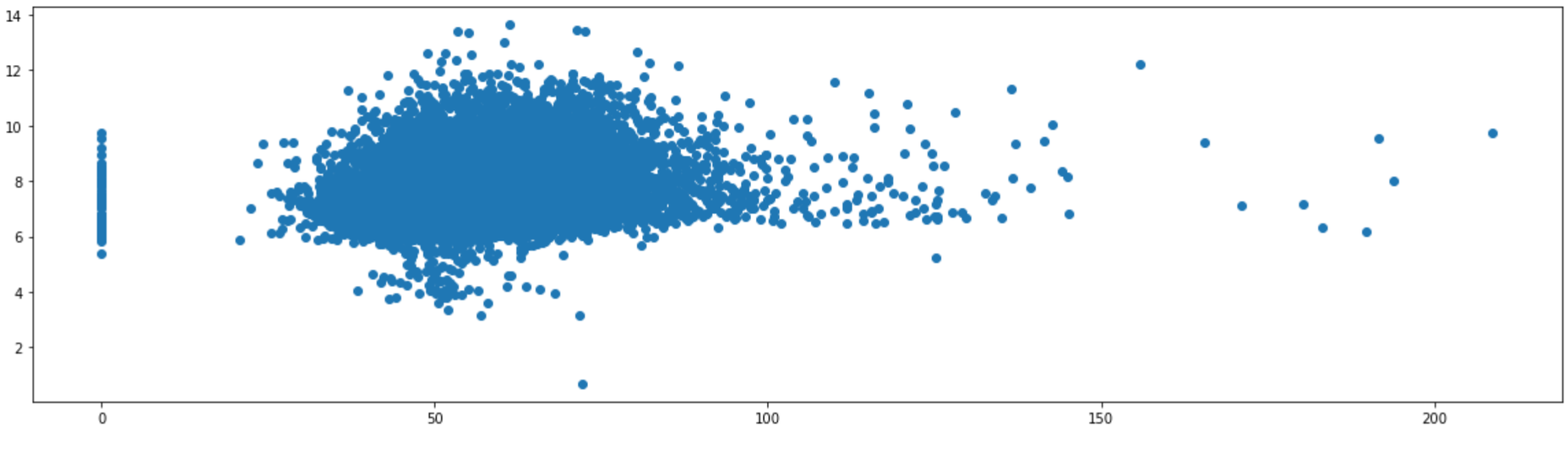
In [409]:
```
1 featureAnalysis('kw_avg_avg',dataset)
```

Out[409]:

	count	mean	std	min	25%	50%	75%	max
kw_avg_avg	34293.0	55.220127	10.130397	0.0	49.038081	53.764534	60.09625	208.728675





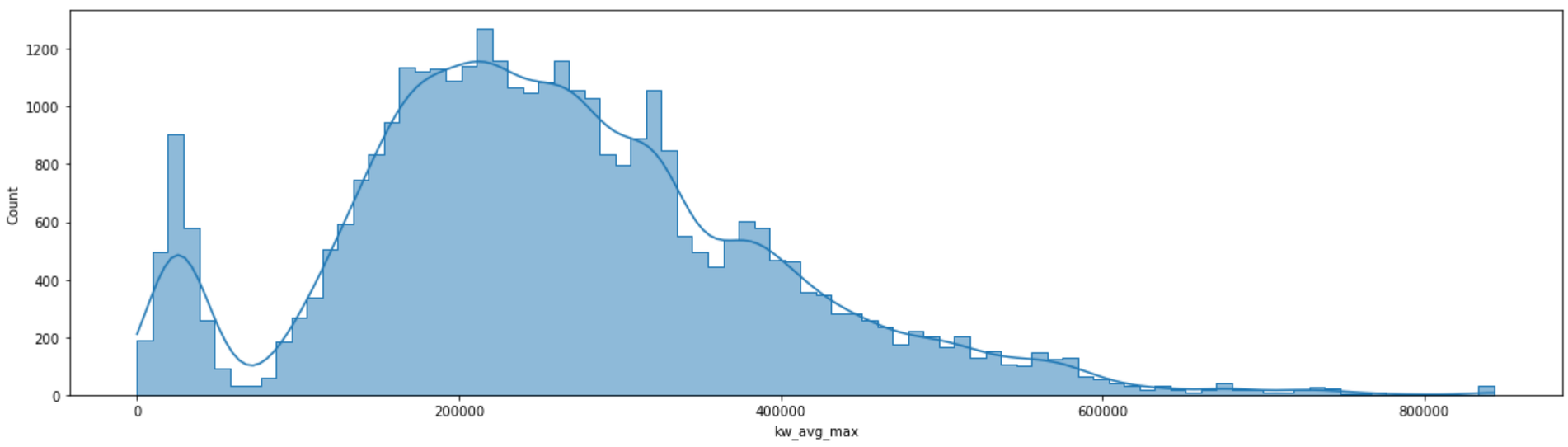


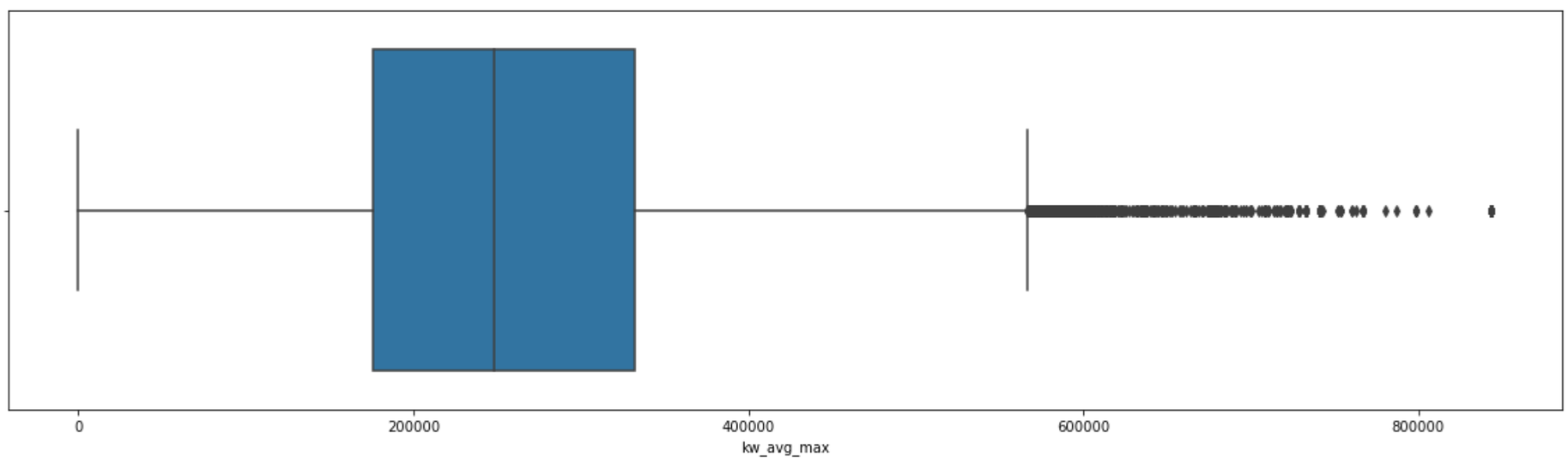
kw_avg_max

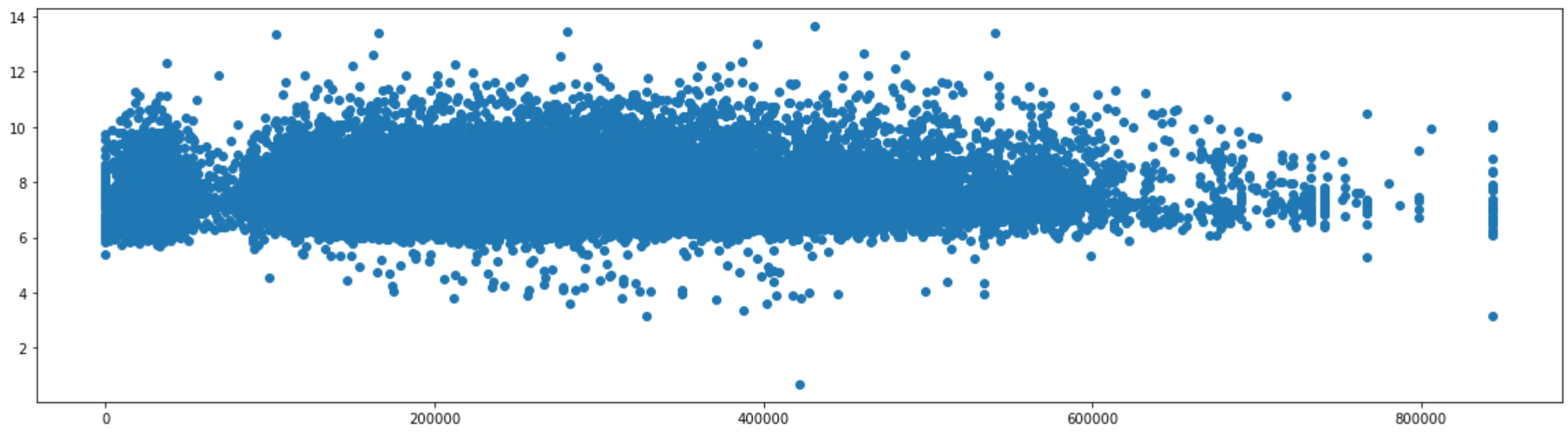
In [410]:
```
1 featureAnalysis('kw_avg_max',dataset)
```

Out[410]:

	count	mean	std	min	25%	50%	75%	max
kw_avg_max	34293.0	261028.633392	131797.390973	0.0	175670.0	248575.0	332066.666667	843300.0





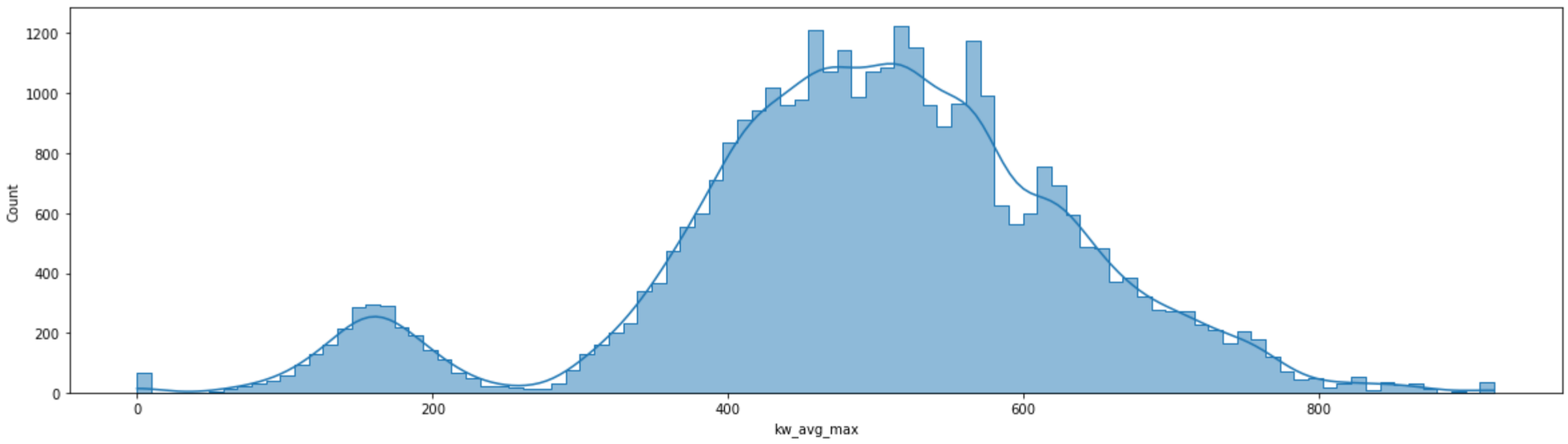


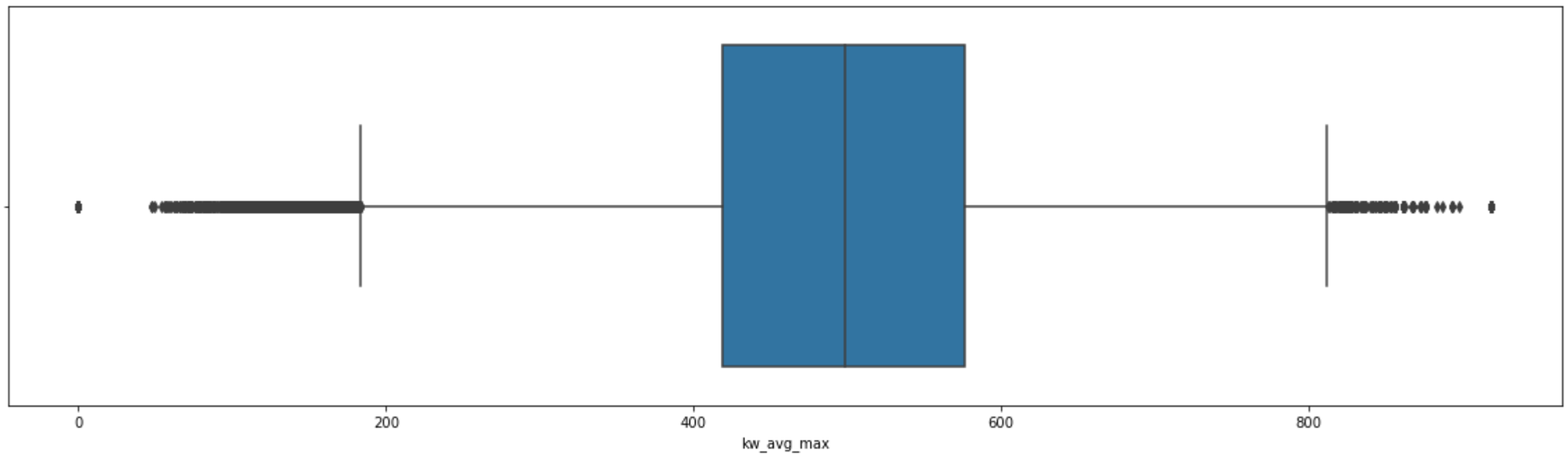
In [411]:
```
1 dataset['kw_avg_max'] = np.sqrt(dataset.kw_avg_max)
```

In [412]:
```
1 featureAnalysis('kw_avg_max',dataset)
```

Out[412]:

	count	mean	std	min	25%	50%	75%	max
kw_avg_max	34293.0	490.968908	141.346197	0.0	419.130051	498.572964	576.25226	918.313672





In [413]:
```
1  dicto = {}
2  def outlierDetection(dataset):
3      num_cols = dataset.select_dtypes(['int64','float64']).columns
4      for column in num_cols:
5          q1 = dataset[column].quantile(0.25)    # First Quartile
6          q3 = dataset[column].quantile(0.75)    # Third Quartile
7          IQR = q3 - q1                            # Inter Quartile Range
8          llimit = q1 - 1.5*IQR                       # Lower Limit
9          ulimit = q3 + 1.5*IQR                        # Upper Limit
10         outliers = dataset[(dataset[column] < llimit) | (dataset[column] > ulimit)]
11         dicto[column] = len(outliers)
12     sorted_dicto = sorted(dicto.items(), key=lambda x:x[1])
13     return (sorted_dicto)
```

In [414]:
```
outlierDetection(dataset)
```

Out[414]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_03', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('media', 0),
 ('neg_pos_min', 24),
 ('num_keywords', 29),
 ('n_tokens_title', 147),
 ('neg_pos_max', 151),
 ('n_unique_tokens', 418),
 ('rate_positive_words', 467),
 ('global_rate_positive_words', 475),
 ('average_token_length', 525),
 ('global_subjectivity', 765),
 ('avg_negative_polarity', 785),
 ('neg_pos_avg', 806),
 ('global_sentiment_polarity', 818),
 ('avg_positive_polarity', 884),
 ('kw_avg_avg', 954),
 ('global_rate_negative_words', 1262),
 ('shares', 1300),
 ('abs_title_sentiment_polarity', 1417),
 ('num_imgs', 1552),
 ('num_self_hrefs', 1637),
 ('kw_max_avg', 1661),
 ('avg_len', 1732),
 ('n_tokens_content', 1770),
 ('kw_avg_min', 1773),
 ('num_videos', 1878),
 ('kw_avg_max', 2182),
 ('max_negative_polarity', 2279),
 ('num_hrefs', 2555),
 ('min_positive_polarity', 2943),
 ('kw_max_min', 3324),
 ('LDA_02', 3517),
 ('self_reference_max_shares', 3545),
 ('self_reference_avg_sharess', 3648),
 ('kw_min_min', 3851),
 ('kw_min_max', 4141),
 ('self_reference_min_shares', 4287),
 ('LDA_00', 4406),
 ('is_weekend', 4575),
 ('LDA_01', 4835),
 ('title_sentiment_polarity', 6591)]
```

In [168]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.6448283721337132
MSE : 0.7461141579054319
RMSE : 0.8637789983007412
r2_score : 0.11725124633813078  12.0

Ridge Model.....................................

The train score for ridge model is 0.11410783663418123
The test score for ridge model is 0.1172178566392219
MAE : 0.6449151479226611
MSE : 0.7461423794429654
r2_score : 0.1172178566392219  12.0

Lasso Model.....................................

The train score for ls model is 0.022033483754844152
The test score for ls model is 0.01783201880637053
MAE : 0.6916114347031985
MSE : 0.8301449683956852
r2_score : 0.01783201880637053  2.0
```

Now it's time to scale our dataset

- Despite all transformation we did in the previous parts there are plenty of outliers.so we are going to use different scaling methods for different features
- Robust scaler for noisy features
- Standardization for normal distribution features
- for others Min_max Scaler(except categoricals)

In [453]:
```
dataset_featureScale = dataset.copy()
```

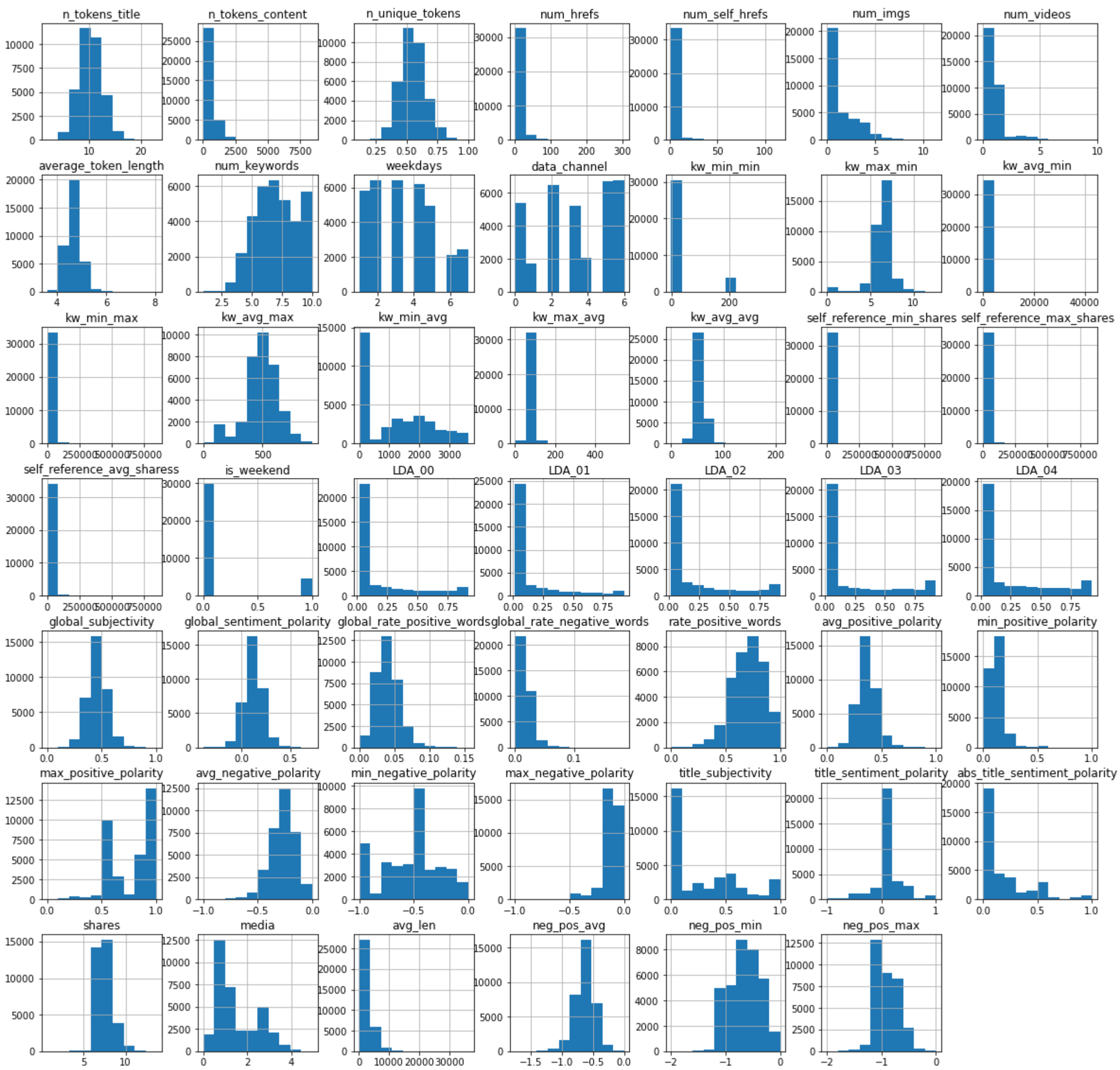
In [454]:
```
dicto = {}
def outlierDetection(dataset):
    num_cols = dataset.select_dtypes(['int64','float64']).columns
    for column in num_cols:
        q1 = dataset[column].quantile(0.25)    # First Quartile
        q3 = dataset[column].quantile(0.75)    # Third Quartile
        IQR = q3 - q1                            # Inter Quartile Range
        llimit = q1 - 1.5*IQR                       # Lower Limit
        ulimit = q3 + 1.5*IQR                        # Upper Limit
        outliers = dataset[(dataset[column] < llimit) | (dataset[column] > ulimit)]
        dicto[column] = len(outliers)
    sorted_dicto = sorted(dicto.items(), key=lambda x:x[1])
    return (sorted_dicto)
```

In [455]:
```
1 outlierDetection(dataset)
```

Out[455]:
```
[('weekdays', 0),
 ('data_channel', 0),
 ('kw_min_avg', 0),
 ('LDA_03', 0),
 ('LDA_04', 0),
 ('max_positive_polarity', 0),
 ('min_negative_polarity', 0),
 ('title_subjectivity', 0),
 ('media', 0),
 ('neg_pos_min', 24),
 ('num_keywords', 29),
 ('n_tokens_title', 147),
 ('neg_pos_max', 151),
 ('n_unique_tokens', 418),
 ('rate_positive_words', 467),
 ('global_rate_positive_words', 475),
 ('average_token_length', 525),
 ('global_subjectivity', 765),
 ('avg_negative_polarity', 785),
 ('neg_pos_avg', 806),
 ('global_sentiment_polarity', 818),
 ('avg_positive_polarity', 884),
 ('kw_avg_avg', 954),
 ('global_rate_negative_words', 1262),
 ('shares', 1300),
 ('abs_title_sentiment_polarity', 1417),
 ('num_imgs', 1552),
 ('num_self_hrefs', 1637),
 ('kw_max_avg', 1661),
 ('avg_len', 1732),
 ('n_tokens_content', 1770),
 ('kw_avg_min', 1773),
 ('num_videos', 1878),
 ('kw_avg_max', 2182),
 ('max_negative_polarity', 2279),
 ('num_hrefs', 2555),
 ('min_positive_polarity', 2943),
 ('kw_max_min', 3324),
 ('LDA_02', 3517),
 ('self_reference_max_shares', 3545),
 ('self_reference_avg_sharess', 3648),
 ('kw_min_min', 3851),
 ('kw_min_max', 4141),
 ('self_reference_min_shares', 4287),
 ('LDA_00', 4406),
 ('is_weekend', 4575),
 ('LDA_01', 4835),
 ('title_sentiment_polarity', 6591)]
```

In [456]:
```
1 dataset.hist(figsize=(20,20))
```

Out[456]:
```
array([[<AxesSubplot:title={'center':'n_tokens_title'}>,
        <AxesSubplot:title={'center':'n_tokens_content'}>,
        <AxesSubplot:title={'center':'n_unique_tokens'}>,
        <AxesSubplot:title={'center':'num_hrefs'}>,
        <AxesSubplot:title={'center':'num_self_hrefs'}>,
        <AxesSubplot:title={'center':'num_imgs'}>,
        <AxesSubplot:title={'center':'num_videos'}>],
       [<AxesSubplot:title={'center':'average_token_length'}>,
        <AxesSubplot:title={'center':'num_keywords'}>,
        <AxesSubplot:title={'center':'weekdays'}>,
        <AxesSubplot:title={'center':'data_channel'}>,
        <AxesSubplot:title={'center':'kw_min_min'}>,
        <AxesSubplot:title={'center':'kw_max_min'}>,
        <AxesSubplot:title={'center':'kw_avg_min'}>],
       [<AxesSubplot:title={'center':'kw_min_max'}>,
        <AxesSubplot:title={'center':'kw_avg_max'}>,
        <AxesSubplot:title={'center':'kw_min_avg'}>,
        <AxesSubplot:title={'center':'kw_max_avg'}>,
        <AxesSubplot:title={'center':'kw_avg_avg'}>,
        <AxesSubplot:title={'center':'self_reference_min_shares'}>,
        <AxesSubplot:title={'center':'self_reference_max_shares'}>],
       [<AxesSubplot:title={'center':'self_reference_avg_sharess'}>,
        <AxesSubplot:title={'center':'is_weekend'}>,
        <AxesSubplot:title={'center':'LDA_00'}>,
        <AxesSubplot:title={'center':'LDA_01'}>,
        <AxesSubplot:title={'center':'LDA_02'}>,
        <AxesSubplot:title={'center':'LDA_03'}>,
        <AxesSubplot:title={'center':'LDA_04'}>],
       [<AxesSubplot:title={'center':'global_subjectivity'}>,
        <AxesSubplot:title={'center':'global_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'global_rate_positive_words'}>,
        <AxesSubplot:title={'center':'global_rate_negative_words'}>,
        <AxesSubplot:title={'center':'rate_positive_words'}>,
        <AxesSubplot:title={'center':'avg_positive_polarity'}>,
        <AxesSubplot:title={'center':'min_positive_polarity'}>],
       [<AxesSubplot:title={'center':'max_positive_polarity'}>,
        <AxesSubplot:title={'center':'avg_negative_polarity'}>,
        <AxesSubplot:title={'center':'min_negative_polarity'}>,
        <AxesSubplot:title={'center':'max_negative_polarity'}>,
        <AxesSubplot:title={'center':'title_subjectivity'}>,
        <AxesSubplot:title={'center':'title_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'abs_title_sentiment_polarity'}>],
       [<AxesSubplot:title={'center':'shares'}>,
        <AxesSubplot:title={'center':'media'}>,
        <AxesSubplot:title={'center':'avg_len'}>,
        <AxesSubplot:title={'center':'neg_pos_avg'}>,
        <AxesSubplot:title={'center':'neg_pos_min'}>,
        <AxesSubplot:title={'center':'neg_pos_max'}>, <AxesSubplot:>]],
      dtype=object)
```

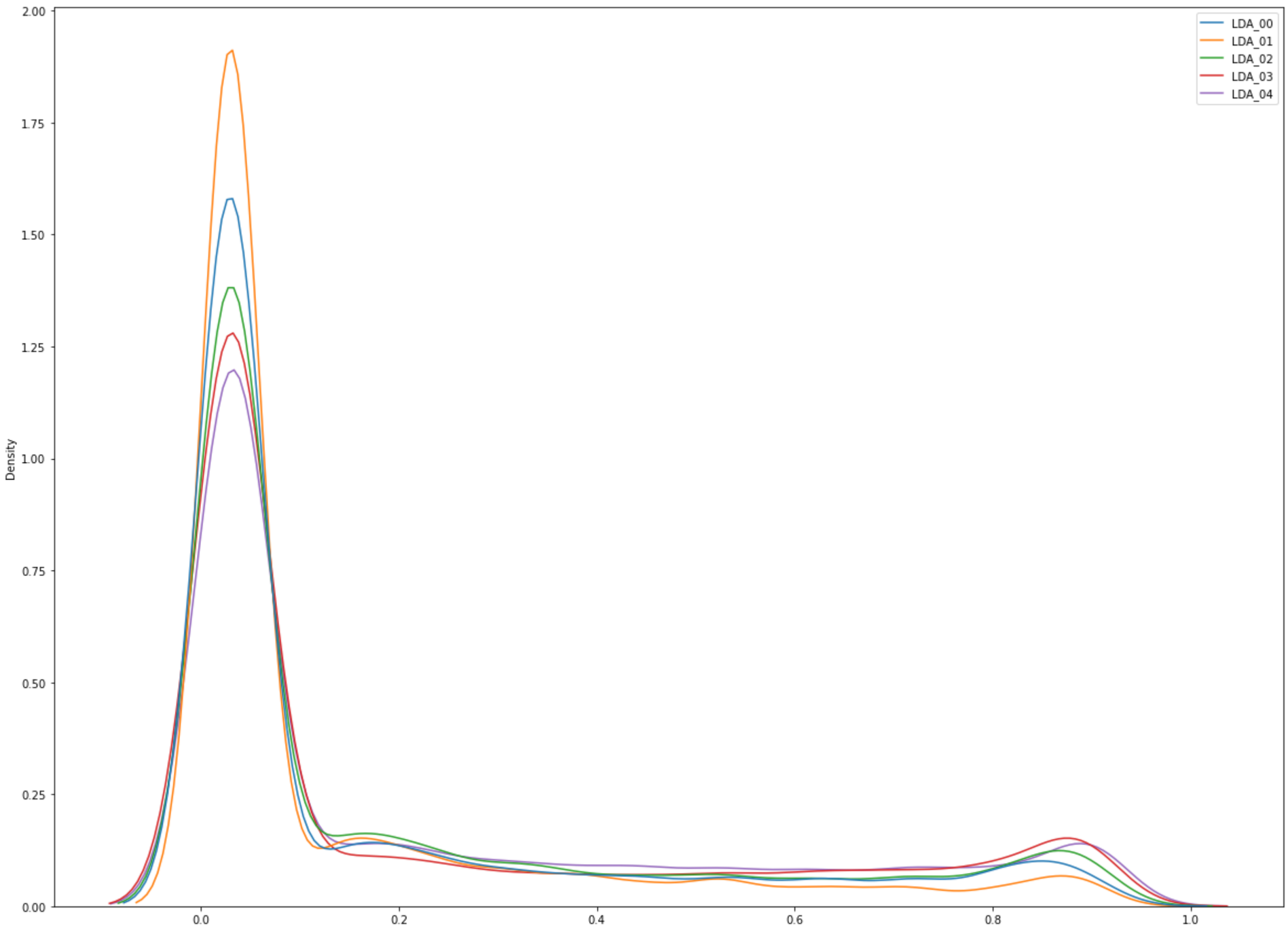


In [457]:
```
1 categorical = ['data_channel' ,'is_weekend' , 'weekdays']
```

In [458]:
```
1 LDA = ['LDA_00', 'LDA_01', 'LDA_02', 'LDA_03', 'LDA_04']
```

In [459]:
```
plt.figure(figsize=(20,15))
sns.kdeplot(data=dataset[LDA])
```

Out[459]: <AxesSubplot:ylabel='Density'>



Using the Robust scaler for features with more than 1300

In [460]:
```
scaler = RobustScaler()
```

In [461]:
```
columnsRobust = []
```

In [462]:
```
for item in dicto:
    if dicto[item] > 1300:
        columnsRobust.append(item)
```

In [463]:
```
columnsRobust
```

Out[463]:
```
['n_tokens_content',
 'num_hrefs',
 'num_self_hrefs',
 'num_imgs',
 'num_videos',
 'kw_min_min',
 'kw_max_min',
 'kw_avg_min',
 'kw_min_max',
 'kw_avg_max',
 'kw_max_avg',
 'self_reference_min_shares',
 'self_reference_max_shares',
 'self_reference_avg_sharess',
 'is_weekend',
 'LDA_00',
 'LDA_01',
 'LDA_02',
 'min_positive_polarity',
 'max_negative_polarity',
 'title_sentiment_polarity',
 'abs_title_sentiment_polarity',
 'avg_len']
```

In [464]:
```
columnsRobust.remove('is_weekend')
```

In [465]:
```
len(columnsRobust)
```

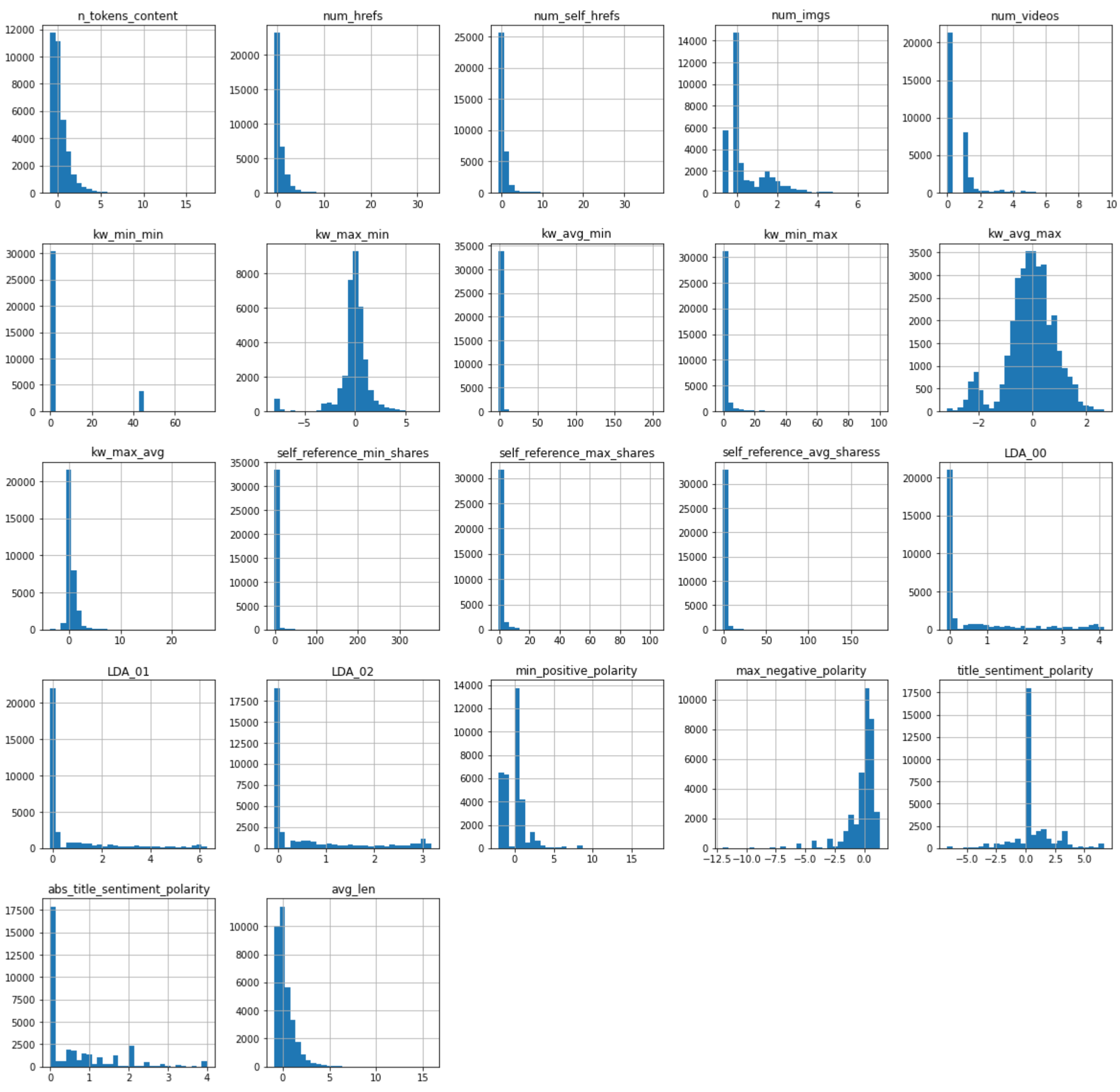
Out[465]: 22

In [466]:
```
scale = RobustScaler().fit(dataset[columnsRobust])
```

In [467]:
```
dataset[columnsRobust] = scale.transform(dataset[columnsRobust])
```

In [468]:
```
1 dataset[columnsRobust].hist(bins=30,figsize=(20,20))
```

Out[468]:
```
array([[<AxesSubplot:title={'center':'n_tokens_content'}>,
        <AxesSubplot:title={'center':'num_hrefs'}>,
        <AxesSubplot:title={'center':'num_self_hrefs'}>,
        <AxesSubplot:title={'center':'num_imgs'}>,
        <AxesSubplot:title={'center':'num_videos'}>],
       [<AxesSubplot:title={'center':'kw_min_min'}>,
        <AxesSubplot:title={'center':'kw_max_min'}>,
        <AxesSubplot:title={'center':'kw_avg_min'}>,
        <AxesSubplot:title={'center':'kw_min_max'}>,
        <AxesSubplot:title={'center':'kw_avg_max'}>],
       [<AxesSubplot:title={'center':'kw_max_avg'}>,
        <AxesSubplot:title={'center':'self_reference_min_shares'}>,
        <AxesSubplot:title={'center':'self_reference_max_shares'}>,
        <AxesSubplot:title={'center':'self_reference_avg_sharess'}>,
        <AxesSubplot:title={'center':'LDA_00'}>],
       [<AxesSubplot:title={'center':'LDA_01'}>,
        <AxesSubplot:title={'center':'LDA_02'}>,
        <AxesSubplot:title={'center':'min_positive_polarity'}>,
        <AxesSubplot:title={'center':'max_negative_polarity'}>,
        <AxesSubplot:title={'center':'title_sentiment_polarity'}>],
       [<AxesSubplot:title={'center':'abs_title_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'avg_len'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



In [469]:
```
1 others = []
```

In [470]:
```
1 for item in dicto:
2     if dicto[item] <= 1300:
3         others.append(item)
```

In [471]:
```
others
```

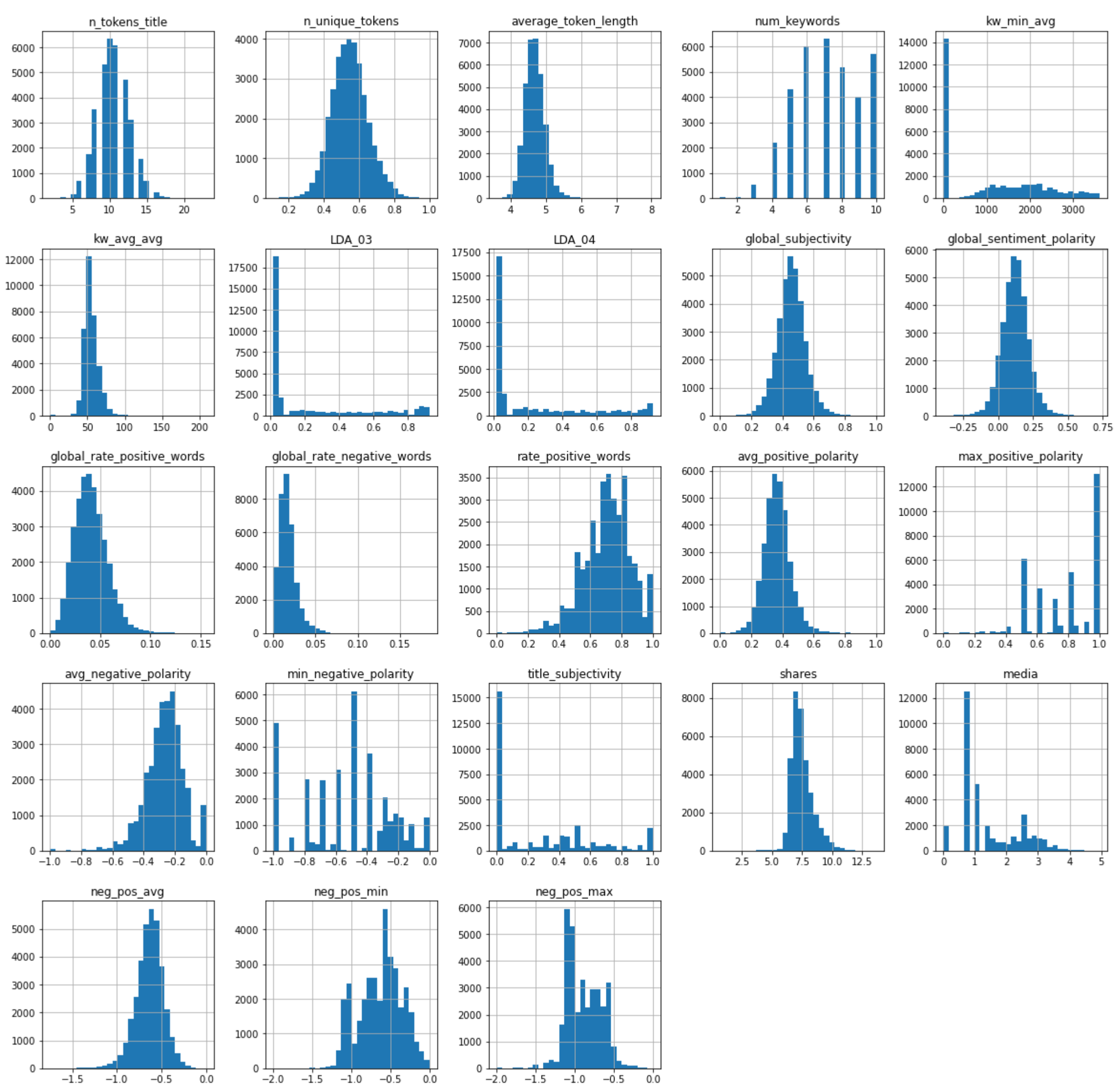
Out[471]:
```
['n_tokens_title',
 'n_unique_tokens',
 'average_token_length',
 'num_keywords',
 'weekdays',
 'data_channel',
 'kw_min_avg',
 'kw_avg_avg',
 'LDA_03',
 'LDA_04',
 'global_subjectivity',
 'global_sentiment_polarity',
 'global_rate_positive_words',
 'global_rate_negative_words',
 'rate_positive_words',
 'avg_positive_polarity',
 'max_positive_polarity',
 'avg_negative_polarity',
 'min_negative_polarity',
 'title_subjectivity',
 'shares',
 'media',
 'neg_pos_avg',
 'neg_pos_min',
 'neg_pos_max']
```

In [472]:
```
others.remove('weekdays')
others.remove('data_channel')
```

In [473]:
```
elseData = dataset[others]
```

In [474]:
```
1 elseData.hist(bins=30,figsize=(20,20))
```

Out[474]:
```
array([[<AxesSubplot:title={'center':'n_tokens_title'}>,
        <AxesSubplot:title={'center':'n_unique_tokens'}>,
        <AxesSubplot:title={'center':'average_token_length'}>,
        <AxesSubplot:title={'center':'num_keywords'}>,
        <AxesSubplot:title={'center':'kw_min_avg'}>],
       [<AxesSubplot:title={'center':'kw_avg_avg'}>,
        <AxesSubplot:title={'center':'LDA_03'}>,
        <AxesSubplot:title={'center':'LDA_04'}>,
        <AxesSubplot:title={'center':'global_subjectivity'}>,
        <AxesSubplot:title={'center':'global_sentiment_polarity'}>],
       [<AxesSubplot:title={'center':'global_rate_positive_words'}>,
        <AxesSubplot:title={'center':'global_rate_negative_words'}>,
        <AxesSubplot:title={'center':'rate_positive_words'}>,
        <AxesSubplot:title={'center':'avg_positive_polarity'}>,
        <AxesSubplot:title={'center':'max_positive_polarity'}>],
       [<AxesSubplot:title={'center':'avg_negative_polarity'}>,
        <AxesSubplot:title={'center':'min_negative_polarity'}>,
        <AxesSubplot:title={'center':'title_subjectivity'}>,
        <AxesSubplot:title={'center':'shares'}>,
        <AxesSubplot:title={'center':'media'}>],
       [<AxesSubplot:title={'center':'neg_pos_avg'}>,
        <AxesSubplot:title={'center':'neg_pos_min'}>,
        <AxesSubplot:title={'center':'neg_pos_max'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```

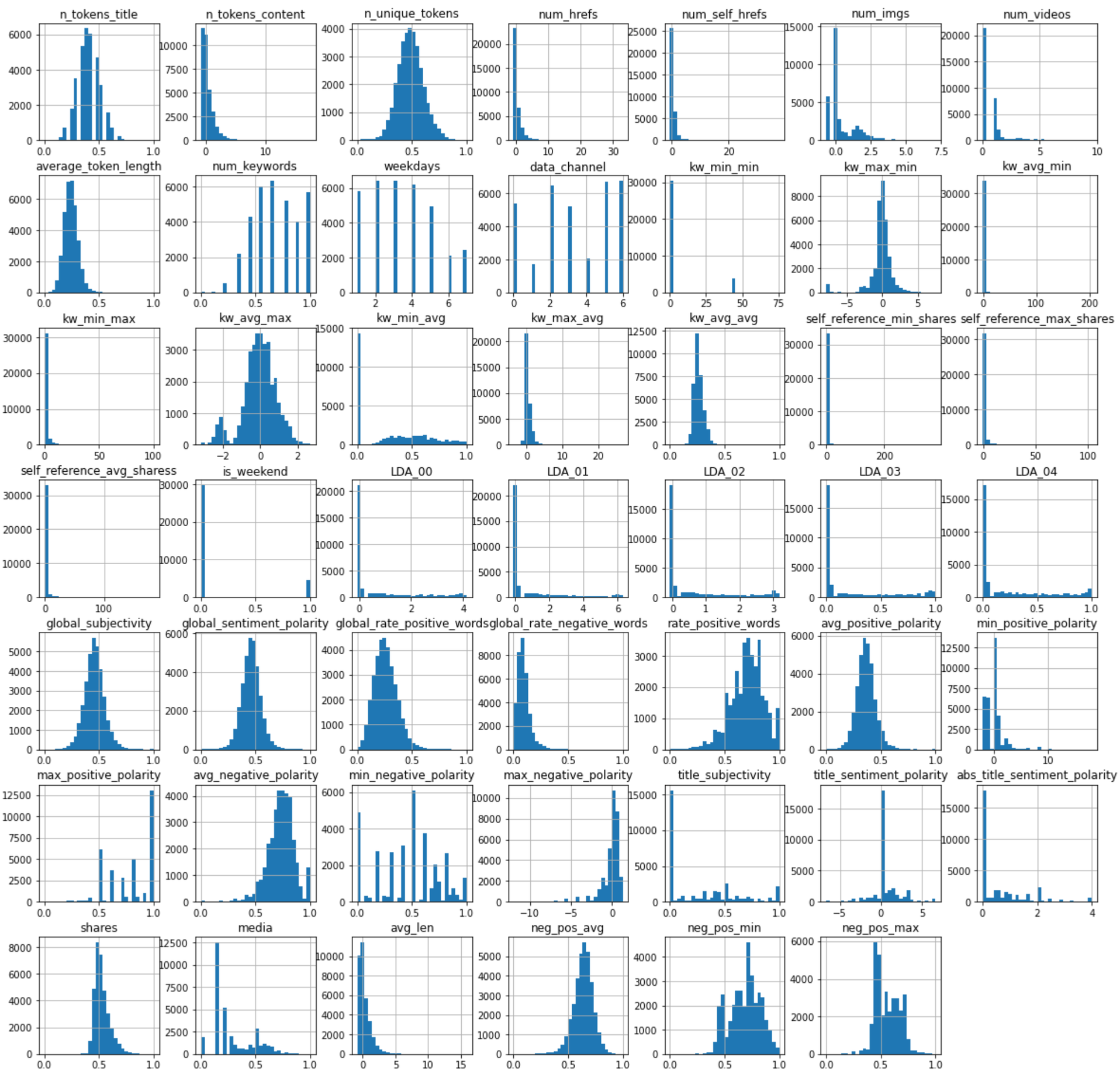


In [475]:
```
1 Normal = MinMaxScaler().fit(dataset[others])
```

In [476]:
```
1 dataset[others] = Normal.transform(dataset[others])
```

In [477]:
```
1 dataset.hist(bins=30,figsize=(20,20))
```

Out[477]:
```
array([[<AxesSubplot:title={'center':'n_tokens_title'}>,
        <AxesSubplot:title={'center':'n_tokens_content'}>,
        <AxesSubplot:title={'center':'n_unique_tokens'}>,
        <AxesSubplot:title={'center':'num_hrefs'}>,
        <AxesSubplot:title={'center':'num_self_hrefs'}>,
        <AxesSubplot:title={'center':'num_imgs'}>,
        <AxesSubplot:title={'center':'num_videos'}>],
       [<AxesSubplot:title={'center':'average_token_length'}>,
        <AxesSubplot:title={'center':'num_keywords'}>,
        <AxesSubplot:title={'center':'weekdays'}>,
        <AxesSubplot:title={'center':'data_channel'}>,
        <AxesSubplot:title={'center':'kw_min_min'}>,
        <AxesSubplot:title={'center':'kw_max_min'}>,
        <AxesSubplot:title={'center':'kw_avg_min'}>],
       [<AxesSubplot:title={'center':'kw_min_max'}>,
        <AxesSubplot:title={'center':'kw_avg_max'}>,
        <AxesSubplot:title={'center':'kw_min_avg'}>,
        <AxesSubplot:title={'center':'kw_max_avg'}>,
        <AxesSubplot:title={'center':'kw_avg_avg'}>,
        <AxesSubplot:title={'center':'self_reference_min_shares'}>,
        <AxesSubplot:title={'center':'self_reference_max_shares'}>],
       [<AxesSubplot:title={'center':'self_reference_avg_sharess'}>,
        <AxesSubplot:title={'center':'is_weekend'}>,
        <AxesSubplot:title={'center':'LDA_00'}>,
        <AxesSubplot:title={'center':'LDA_01'}>,
        <AxesSubplot:title={'center':'LDA_02'}>,
        <AxesSubplot:title={'center':'LDA_03'}>,
        <AxesSubplot:title={'center':'LDA_04'}>],
       [<AxesSubplot:title={'center':'global_subjectivity'}>,
        <AxesSubplot:title={'center':'global_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'global_rate_positive_words'}>,
        <AxesSubplot:title={'center':'global_rate_negative_words'}>,
        <AxesSubplot:title={'center':'rate_positive_words'}>,
        <AxesSubplot:title={'center':'avg_positive_polarity'}>,
        <AxesSubplot:title={'center':'min_positive_polarity'}>],
       [<AxesSubplot:title={'center':'max_positive_polarity'}>,
        <AxesSubplot:title={'center':'avg_negative_polarity'}>,
        <AxesSubplot:title={'center':'min_negative_polarity'}>,
        <AxesSubplot:title={'center':'max_negative_polarity'}>,
        <AxesSubplot:title={'center':'title_subjectivity'}>,
        <AxesSubplot:title={'center':'title_sentiment_polarity'}>,
        <AxesSubplot:title={'center':'abs_title_sentiment_polarity'}>],
       [<AxesSubplot:title={'center':'shares'}>,
        <AxesSubplot:title={'center':'media'}>,
        <AxesSubplot:title={'center':'avg_len'}>,
        <AxesSubplot:title={'center':'neg_pos_avg'}>,
        <AxesSubplot:title={'center':'neg_pos_min'}>,
        <AxesSubplot:title={'center':'neg_pos_max'}>, <AxesSubplot:>]],
      dtype=object)
```

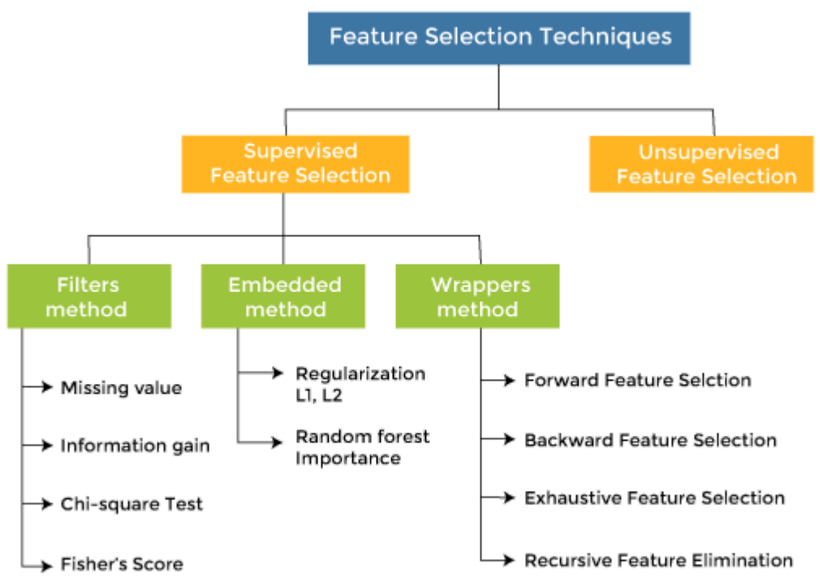


In [478]:
```
simpleLinear(dataset)
ridge_regression(dataset,10)
lasso_regression(dataset,10)
```

```
Linear Model.....................................

MAE : 0.049787095087776435
MSE : 0.004447808465286461
RMSE : 0.06669189205058185
r2_score : 0.11723110156833583  12.0

Ridge Model.....................................

The train score for ridge model is 0.10695438696269477
The test score for ridge model is 0.10963325071325125
MAE : 0.050119141400555164
MSE : 0.004486090042051646
r2_score : 0.10963325071325125  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

4.Apply the feature selection methods that you have implemented in the above sections.

Well we have already dropped two non-predictive feature (url and timedelta)



In [201]:
```
dataset
```

Out[201]:

	n_tokens_title	n_tokens_content	n_unique_tokens	num_hrefs	num_self_hrefs	num_imgs	num_videos	average_token_length	num_keywords	weekdays	data_channel	kw_min_min	kw_max_min	kw_avg_min	kw_
0	0.476190	-0.407328	0.619896	-0.333333	-0.333333	0.000000	0.000000	0.243242	0.444444	1	2	0.2	-7.446974	-1.105258	-
1	0.333333	-0.329741	0.553400	-0.444444	-0.666667	0.000000	0.000000	0.295782	0.333333	1	3	0.2	-7.446974	-1.105258	-
2	0.333333	-0.424569	0.519940	-0.444444	-0.666667	0.000000	0.000000	0.178624	0.555556	1	3	0.2	913.589588	-1.105258	-
4	0.523810	1.431034	0.339740	1.333333	5.333333	2.395419	0.000000	0.243798	0.666667	1	5	0.2	508.562329	-1.105258	-
5	0.380952	-0.081897	0.502719	-0.555556	-0.333333	-0.689898	0.000000	0.170990	0.888889	1	5	0.2	390.715415	-1.105258	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
39639	0.428571	-0.133621	0.467877	0.222222	1.333333	0.000000	1.000000	0.207838	0.777778	3	5	0.0	0.000000	-0.276115	
39640	0.476190	-0.172414	0.656846	0.222222	1.333333	0.505040	6.928203	0.181354	0.666667	3	4	0.0	-0.097850	-0.224032	
39641	0.380952	0.073276	0.453531	1.888889	-0.666667	1.699979	1.000000	0.332525	0.777778	3	0	0.0	0.033604	-0.299463	
39642	0.190476	0.590517	0.479675	0.333333	-0.666667	0.000000	0.000000	0.309594	0.444444	3	6	0.0	1108.094203	-1.110047	-
39643	0.380952	-0.540948	0.663276	-0.666667	-0.666667	-0.689898	1.414214	0.196179	0.333333	3	2	0.0	-2.212843	-0.992710	2

34293 rows × 48 columns

1. Wrappers methods

code:- Exercise 10

forward selection

In [223]:
```
#10 features
```

In [224]:
```
scratch10_forward = forward_selection(dataset,10)
sfs10 = Forward_SFS(dataset,10)
```

In [225]:
```
scratch10_forward.append('shares')
sfs10.append('shares')
```

In [226]:
```
simpleLinear(dataset[scratch10_forward])
ridge_regression(dataset[scratch10_forward],10)
lasso_regression(dataset[scratch10_forward],10)
```

```
Linear Model.....................................

MAE : 0.05022320423976347
MSE : 0.004496271738988787
RMSE : 0.06705424474996931
r2_score : 0.10761246104581967  11.0

Ridge Model.....................................

The train score for ridge model is 0.0975455984328214
The test score for ridge model is 0.10538692550654949
MAE : 0.05033278840954008
MSE : 0.004507485042752601
r2_score : 0.10538692550654949  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [227]:
```
simpleLinear(dataset[sfs10])
ridge_regression(dataset[sfs10],10)
lasso_regression(dataset[sfs10],10)
```

```
Linear Model.....................................

MAE : 0.05021964157480449
MSE : 0.004514142504966904
RMSE : 0.06718736864148576
r2_score : 0.10406560049196345  10.0

Ridge Model.....................................

The train score for ridge model is 0.09889315602284288
The test score for ridge model is 0.10271664117516477
MAE : 0.05030366606818933
MSE : 0.00452093920190451
r2_score : 0.10271664117516477  10.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [228]:
```
#20 features
```

In [229]:
```
scratch20_forward = forward_selection(dataset,20)
sfs20 = Forward_SFS(dataset,20)
```

In [230]:
```
scratch20_forward.append('shares')
sfs20.append('shares')
```

In [231]:
```
simpleLinear(dataset[scratch20_forward])
ridge_regression(dataset[scratch20_forward],10)
lasso_regression(dataset[scratch20_forward],10)
```

```
Linear Model.....................................

MAE : 0.04987858062894027
MSE : 0.0044512621897896355
RMSE : 0.0667177801623348
r2_score : 0.11654563127460615  12.0

Ridge Model.....................................

The train score for ridge model is 0.10130959240532167
The test score for ridge model is 0.10925385472947424
MAE : 0.050255254563513256
MSE : 0.004488001618990227
r2_score : 0.10925385472947424  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [232]:
```
simpleLinear(dataset[sfs20])
ridge_regression(dataset[sfs20],10)
lasso_regression(dataset[sfs20],10)
```

```
Linear Model.....................................

MAE : 0.049820187518791234
MSE : 0.004451693257842113
RMSE : 0.0667210106176616
r2_score : 0.1164600760010811  12.0

Ridge Model.....................................

The train score for ridge model is 0.10461740862919855
The test score for ridge model is 0.10928532398253654
MAE : 0.050165662755702466
MSE : 0.0044878430619653764
r2_score : 0.10928532398253654  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [233]:
```
#30 features
```

In [234]:
```
scratch30_forward = forward_selection(dataset,30)
sfs30 = Forward_SFS(dataset,30)
```

In [235]:
```
scratch30_forward.append('shares')
sfs30.append('shares')
```

In [236]:
```
simpleLinear(dataset[scratch30_forward])
ridge_regression(dataset[scratch30_forward],10)
lasso_regression(dataset[scratch30_forward],10)
```

```
Linear Model.....................................

MAE : 0.049780676306150504
MSE : 0.004444523411480211
RMSE : 0.06666725891680421
r2_score : 0.11788309532041963  12.0

Ridge Model.....................................

The train score for ridge model is 0.1049787619301843
The test score for ridge model is 0.10963554826823285
MAE : 0.050152668692952415
MSE : 0.004486078465879768
r2_score : 0.10963554826823285  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [237]:
```
simpleLinear(dataset[sfs30])
ridge_regression(dataset[sfs30],10)
lasso_regression(dataset[sfs30],10)
```

```
Linear Model.....................................

MAE : 0.04979455111328965
MSE : 0.0044459301616880335
RMSE : 0.06667780861492101
r2_score : 0.11760389369988677  12.0

Ridge Model.....................................

The train score for ridge model is 0.10654595696752833
The test score for ridge model is 0.1094601173842299
MAE : 0.05015236459158638
MSE : 0.00448696236989171
r2_score : 0.1094601173842299  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [238]:
```
#40 features
```

In [239]:
```
scratch40_forward = forward_selection(dataset,40)
sfs40 = Forward_SFS(dataset,40)
```

In [240]:
```
scratch40_forward.append('shares')
sfs40.append('shares')
```

In [241]:
```
simpleLinear(dataset[scratch40_forward])
ridge_regression(dataset[scratch40_forward],10)
lasso_regression(dataset[scratch40_forward],10)
```

```
Linear Model.....................................

MAE : 0.04977027568003537
MSE : 0.004444653579904456
RMSE : 0.06666823516416537
r2_score : 0.1178572604317597  12.0

Ridge Model.....................................

The train score for ridge model is 0.1057912024428691
The test score for ridge model is 0.11017148571926216
MAE : 0.050121815936052756
MSE : 0.0044833781587713195
r2_score : 0.11017148571926216  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [242]:
```
simpleLinear(dataset[sfs40])
ridge_regression(dataset[sfs40],10)
lasso_regression(dataset[sfs40],10)
```

```
Linear Model.....................................

MAE : 0.04978448429933722
MSE : 0.004447385907268924
RMSE : 0.06668872398890928
r2_score : 0.11731496783159245  12.0

Ridge Model.....................................

The train score for ridge model is 0.10695344863354495
The test score for ridge model is 0.10963774856287523
MAE : 0.05011922641891463
MSE : 0.004486067379751616
r2_score : 0.10963774856287523  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

Polynomial Regression

In [243]:
```
#degree = 2
```

In [244]:
```
polynomial(2,dataset[scratch40_forward])
```

```
Polynomial Model.....................................

MAE : 0.04868348746777315
MSE : 0.0043910541684915125
RMSE : 0.06626502975545633
r2_score : 0.12849528446963654  13.0
```

In [245]:
```
polynomial(2,dataset[scratch30_forward])
```

```
Polynomial Model.....................................

MAE : 0.048559386648008254
MSE : 0.004436327170017757
RMSE : 0.06660575928564855
r2_score : 0.11950982612583982  12.0
```

In [246]:
```
polynomial(2,dataset[scratch20_forward])
```

```
Polynomial Model.....................................

MAE : 0.048774347285023806
MSE : 0.004427818030827651
RMSE : 0.06654185172376594
r2_score : 0.12119865861223789  12.0
```

In [247]:
```
polynomial(2,dataset[scratch10_forward])
```

```
Polynomial Model.....................................

MAE : 0.049256186584902426
MSE : 0.004404655394189925
RMSE : 0.06636757788400843
r2_score : 0.1257958114323383  13.0
```

In [248]:
```
#degree = 3
```

In [249]:
```
polynomial(3,dataset[scratch30_forward])
```

```
Polynomial Model.....................................

MAE : 0.07001423139947488
MSE : 0.2669372153965609
RMSE : 0.5166596707665123
r2_score : -51.9797704701435  -5198.0
```

In [250]:
```
polynomial(3,dataset[scratch20_forward])
```

```
Polynomial Model.....................................

MAE : 0.05377163794192986
MSE : 0.034853462384816424
RMSE : 0.1866908203014182
r2_score : -5.917463473551856  -592.0
```

In [251]:
```
polynomial(3,dataset[scratch10_forward])
```

```
Polynomial Model.....................................

MAE : 0.04926228076754164
MSE : 0.0044308216655263
RMSE : 0.06656441741295645
r2_score : 0.1206025188919887  12.0
```

In [252]:
```
#degree = 4
```

In [253]:
```
polynomial(3,dataset[scratch10_forward])
```

```
Polynomial Model.....................................

MAE : 0.04926228076754164
MSE : 0.0044308216655263
RMSE : 0.06656441741295645
r2_score : 0.1206025188919887  12.0
```

Backward selcetion

In [216]:
```
#10 features
```

In [498]:
```
scratch10_backward = backwardSelection(dataset, 10)
sbs10 = Backward_SBS(dataset,10)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  47 out of  47 | elapsed:    6.8s finished
Features: 46/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  46 out of  46 | elapsed:    6.6s finished
Features: 45/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    6.4s finished
Features: 44/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  44 out of  44 | elapsed:    6.0s finished
Features: 43/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  43 out of  43 | elapsed:    5.9s finished
Features: 42/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    5.5s finished
Features: 41/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  41 out of  41 | elapsed:    5.1s finished
Features: 40/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    5.0s finished
Features: 39/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  39 out of  39 | elapsed:    4.7s finished
Features: 38/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  38 out of  38 | elapsed:    4.5s finished
Features: 37/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  37 out of  37 | elapsed:    4.2s finished
Features: 36/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed:    4.2s finished
Features: 35/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    3.9s finished
Features: 34/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  34 out of  34 | elapsed:    3.7s finished
Features: 33/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  33 out of  33 | elapsed:    3.5s finished
Features: 32/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  32 out of  32 | elapsed:    3.2s finished
Features: 31/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  31 out of  31 | elapsed:    3.3s finished
Features: 30/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    3.0s finished
Features: 29/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  29 out of  29 | elapsed:    2.8s finished
Features: 28/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:    2.6s finished
Features: 27/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    2.5s finished
Features: 26/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  26 out of  26 | elapsed:    2.3s finished
Features: 25/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    2.1s finished
Features: 24/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  24 out of  24 | elapsed:    2.0s finished
Features: 23/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  23 out of  23 | elapsed:    1.8s finished
Features: 22/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  22 out of  22 | elapsed:    1.7s finished
Features: 21/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed:    1.5s finished
Features: 20/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    1.3s finished
Features: 19/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:    1.3s finished
Features: 18/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:    1.1s finished
Features: 17/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:    1.0s finished
Features: 16/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    0.9s finished
Features: 15/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    0.8s finished
Features: 14/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    0.7s finished
Features: 13/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.6s finished
Features: 12/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.5s finished
Features: 11/10[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.4s finished
Features: 10/10
```

In [502]:
```
scratch10_backward.append('shares')
sbs10.append('shares')
```

In [505]:
```
scratch10_backward
```

Out[505]:
```
['self_reference_min_shares',
 'media',
 'num_keywords',
 'LDA_00',
 'kw_max_avg',
 'kw_min_avg',
 'kw_min_min',
 'is_weekend',
 'num_hrefs',
 'LDA_04',
 'shares']
```

In [506]:
```
sbs10
```

Out[506]:
```
['kw_avg_max',
 'kw_min_avg',
 'kw_max_avg',
 'kw_avg_avg',
 'self_reference_avg_sharess',
 'is_weekend',
 'LDA_00',
 'LDA_04',
 'global_subjectivity',
 'media',
 'shares']
```

In [503]:
```
simpleLinear(dataset[scratch10_backward])
ridge_regression(dataset[scratch10_backward],10)
lasso_regression(dataset[scratch10_backward],10)
```

```
Linear Model.....................................

MAE : 0.05099087060934361
MSE : 0.004609567767511667
RMSE : 0.06789379771018607
r2_score : 0.08512628362239871  9.0

Ridge Model.....................................

The train score for ridge model is 0.07609211515591197
The test score for ridge model is 0.08511701699053231
MAE : 0.050992479567697216
MSE : 0.004609614457198779
r2_score : 0.08511701699053231  9.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [504]:
```
simpleLinear(dataset[sbs10])
ridge_regression(dataset[sbs10],10)
lasso_regression(dataset[sbs10],10)
```

```
Linear Model.....................................

MAE : 0.050152535291464136
MSE : 0.0045086151050355685
RMSE : 0.067146221822494
r2_score : 0.10516263890687239  11.0

Ridge Model.....................................

The train score for ridge model is 0.09657385415469744
The test score for ridge model is 0.09742949824791025
MAE : 0.05047679456290594
MSE : 0.004547578336009485
r2_score : 0.09742949824791025  10.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [221]:
```
#20 features
```

In [507]:
```
scratch20_backward = backwardSelection(dataset, 20)
sbs20 = Backward_SBS(dataset,20)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  47 out of  47 | elapsed:    7.5s finished
Features: 46/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  46 out of  46 | elapsed:    7.4s finished
Features: 45/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    7.3s finished
Features: 44/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  44 out of  44 | elapsed:    7.1s finished
Features: 43/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  43 out of  43 | elapsed:    6.3s finished
Features: 42/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    5.8s finished
Features: 41/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  41 out of  41 | elapsed:    5.4s finished
Features: 40/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    5.2s finished
Features: 39/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  39 out of  39 | elapsed:    4.9s finished
Features: 38/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  38 out of  38 | elapsed:    4.7s finished
Features: 37/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  37 out of  37 | elapsed:    4.4s finished
Features: 36/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed:    4.2s finished
Features: 35/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    4.0s finished
Features: 34/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  34 out of  34 | elapsed:    3.8s finished
Features: 33/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  33 out of  33 | elapsed:    3.5s finished
Features: 32/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  32 out of  32 | elapsed:    3.3s finished
Features: 31/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  31 out of  31 | elapsed:    3.5s finished
Features: 30/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    3.2s finished
Features: 29/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  29 out of  29 | elapsed:    2.9s finished
Features: 28/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:    2.7s finished
Features: 27/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    2.5s finished
Features: 26/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  26 out of  26 | elapsed:    2.3s finished
Features: 25/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    2.1s finished
Features: 24/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  24 out of  24 | elapsed:    2.0s finished
Features: 23/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  23 out of  23 | elapsed:    1.8s finished
Features: 22/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  22 out of  22 | elapsed:    1.7s finished
Features: 21/20[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed:    1.7s finished
Features: 20/20
```

In [508]:
```
scratch20_backward.append('shares')
sbs20.append('shares')
```

In [509]:
```
simpleLinear(dataset[scratch20_backward])
ridge_regression(dataset[scratch20_backward],10)
lasso_regression(dataset[scratch20_backward],10)
```

```
Linear Model.....................................

MAE : 0.050734279112021455
MSE : 0.004564071989642118
RMSE : 0.06755791581777902
r2_score : 0.09415595700573931  9.0

Ridge Model.....................................

The train score for ridge model is 0.08955958928622221
The test score for ridge model is 0.09423506818340033
MAE : 0.05073871232004646
MSE : 0.004563673390001459
r2_score : 0.09423506818340033  9.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [510]:
```
simpleLinear(dataset[sbs20])
ridge_regression(dataset[sbs20],10)
lasso_regression(dataset[sbs20],10)
```

```
Linear Model.....................................

MAE : 0.04984469182918391
MSE : 0.0044605122493864054
RMSE : 0.06678706648286332
r2_score : 0.11470974625743469  11.0

Ridge Model.....................................

The train score for ridge model is 0.10342479432207108
The test score for ridge model is 0.10693048346954326
MAE : 0.05020481573549615
MSE : 0.004499707865524607
r2_score : 0.10693048346954326  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [226]:
```
#30 features
```

In [511]:
```
scratch30_backward = backwardSelection(dataset, 30)
sbs30 = Backward_SBS(dataset,30)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  47 out of  47 | elapsed:    7.2s finished
Features: 46/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  46 out of  46 | elapsed:    6.9s finished
Features: 45/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    6.6s finished
Features: 44/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  44 out of  44 | elapsed:    6.3s finished
Features: 43/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  43 out of  43 | elapsed:    6.0s finished
Features: 42/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    5.8s finished
Features: 41/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  41 out of  41 | elapsed:    5.4s finished
Features: 40/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    5.2s finished
Features: 39/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  39 out of  39 | elapsed:    4.9s finished
Features: 38/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  38 out of  38 | elapsed:    4.7s finished
Features: 37/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  37 out of  37 | elapsed:    4.4s finished
Features: 36/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed:    4.4s finished
Features: 35/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    3.9s finished
Features: 34/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  34 out of  34 | elapsed:    3.8s finished
Features: 33/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  33 out of  33 | elapsed:    3.6s finished
Features: 32/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  32 out of  32 | elapsed:    3.3s finished
Features: 31/30[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  31 out of  31 | elapsed:    3.3s finished
Features: 30/30
```

In [512]:
```
scratch30_backward.append('shares')
sbs30.append('shares')
```

In [513]:
```
simpleLinear(dataset[scratch30_backward])
ridge_regression(dataset[scratch30_backward],10)
lasso_regression(dataset[scratch30_backward],10)
```

```
Linear Model.....................................

MAE : 0.0505659755980215
MSE : 0.004550877129891829
RMSE : 0.06746018922217628
r2_score : 0.09677477746476548  10.0

Ridge Model.....................................

The train score for ridge model is 0.0940630961903901
The test score for ridge model is 0.0968180755870599
MAE : 0.050573080709110846
MSE : 0.0045506589734125805
r2_score : 0.0968180755870599  10.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [514]:
```
simpleLinear(dataset[sbs30])
ridge_regression(dataset[sbs30],10)
lasso_regression(dataset[sbs30],10)
```

```
Linear Model.....................................

MAE : 0.049837192088662004
MSE : 0.004458267952591268
RMSE : 0.06677026248706282
r2_score : 0.11515517807516218  12.0

Ridge Model.....................................

The train score for ridge model is 0.10388069252601129
The test score for ridge model is 0.10734037165339783
MAE : 0.05019299329458933
MSE : 0.004497642654417592
r2_score : 0.10734037165339783  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [231]:
```
#40 features
```

In [515]:
```
scratch40_backward = backwardSelection(dataset, 40)
sbs40 = Backward_SBS(dataset,40)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  47 out of  47 | elapsed:    7.2s finished
Features: 46/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  46 out of  46 | elapsed:    6.9s finished
Features: 45/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    6.6s finished
Features: 44/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  44 out of  44 | elapsed:    6.3s finished
Features: 43/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  43 out of  43 | elapsed:    6.0s finished
Features: 42/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    5.7s finished
Features: 41/40[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  41 out of  41 | elapsed:    5.4s finished
Features: 40/40
```

In [516]:
```
scratch40_backward.append('shares')
sbs40.append('shares')
```

In [517]:
```
simpleLinear(dataset[scratch40_backward])
ridge_regression(dataset[scratch40_backward],10)
lasso_regression(dataset[scratch40_backward],10)
```

```
Linear Model.....................................

MAE : 0.050571606086448395
MSE : 0.004551775628434176
RMSE : 0.06746684836595064
r2_score : 0.09659644996378147  10.0

Ridge Model.....................................

The train score for ridge model is 0.0941063767643211
The test score for ridge model is 0.0966861506429294
MAE : 0.05057863988116857
MSE : 0.004551323673861734
r2_score : 0.0966861506429294  10.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

In [518]:
```
simpleLinear(dataset[sbs40])
ridge_regression(dataset[sbs40],10)
lasso_regression(dataset[sbs40],10)
```

```
Linear Model.....................................

MAE : 0.04984592477635207
MSE : 0.004461873247387708
RMSE : 0.06679725478930783
r2_score : 0.11443962520439144  11.0

Ridge Model.....................................

The train score for ridge model is 0.10484465451523461
The test score for ridge model is 0.10622007242669251
MAE : 0.05020268717233091
MSE : 0.004503287253352881
r2_score : 0.10622007242669251  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0
```

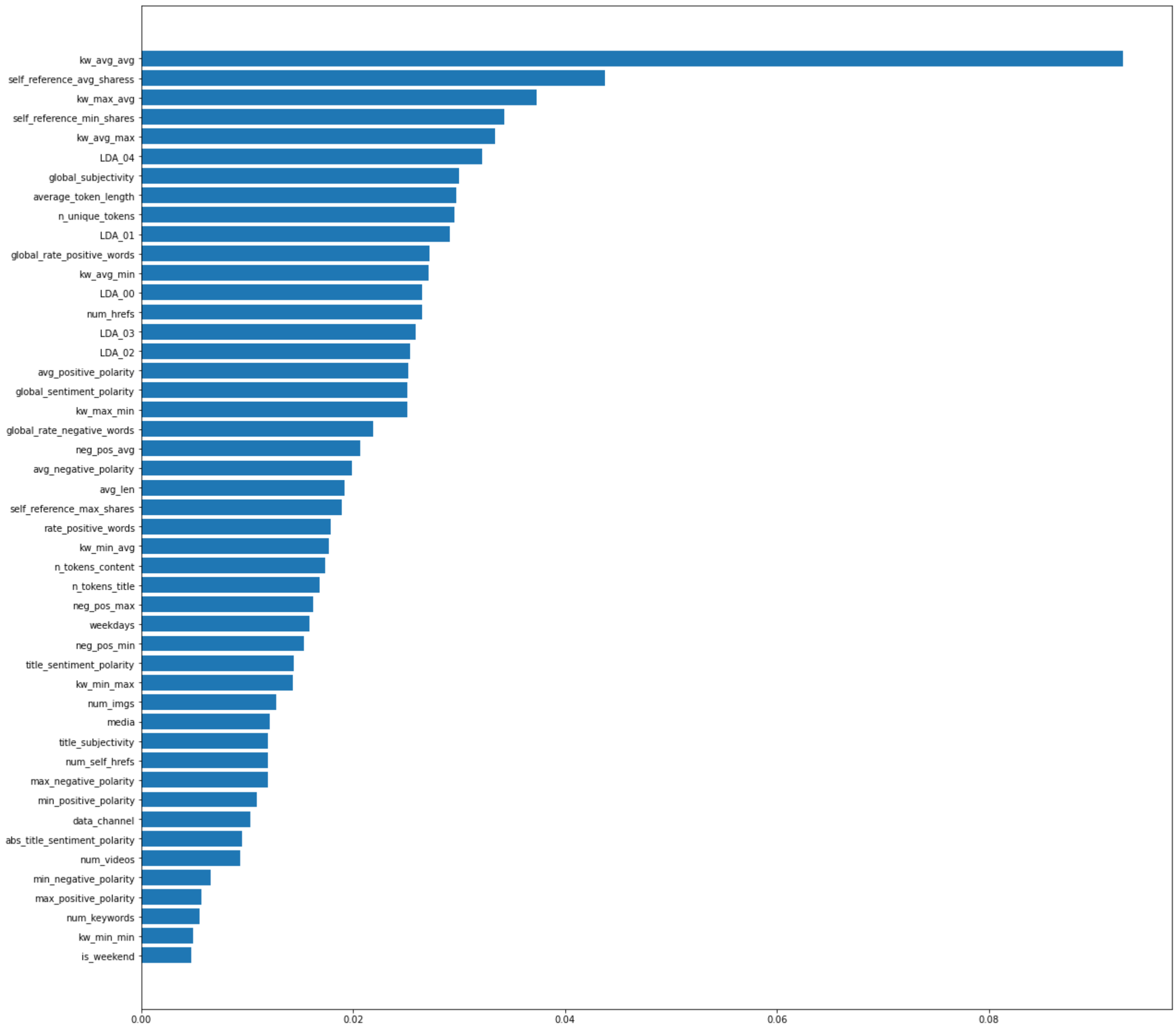
Tree based method Feature importance

1.Random Forest Built-in Feature Importance

In [205]:
```
def informationGain(dataset,n):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    rf = RandomForestRegressor(n_estimators=n)
    rf.fit(X,  Y)
    sorted_idx = rf.feature_importances_.argsort()
    plt.figure(figsize=(20,20))
    plt.barh(X.columns[sorted_idx], rf.feature_importances_[sorted_idx])
    return dataset.columns[sorted_idx]
```

In [206]:
```
1 informationGain(dataset,100)
```

Out[206]:
```
Index(['kw_avg_max', 'LDA_00', 'min_negative_polarity', 'LDA_02', 'LDA_04',
       'title_subjectivity', 'num_imgs', 'data_channel', 'global_subjectivity',
       'LDA_01', 'max_negative_polarity', 'neg_pos_avg', 'LDA_03',
       'avg_negative_polarity', 'is_weekend', 'avg_len',
       'min_positive_polarity', 'neg_pos_min', 'avg_positive_polarity',
       'global_rate_positive_words', 'global_sentiment_polarity',
       'self_reference_avg_sharess', 'title_sentiment_polarity', 'shares',
       'average_token_length', 'num_keywords', 'rate_positive_words',
       'kw_min_min', 'self_reference_max_shares', 'kw_avg_min', 'weekdays',
       'n_unique_tokens', 'num_hrefs', 'max_positive_polarity',
       'n_tokens_title', 'kw_avg_avg', 'kw_max_min', 'n_tokens_content',
       'global_rate_negative_words', 'num_videos', 'kw_min_max',
       'num_self_hrefs', 'kw_max_avg', 'media', 'self_reference_min_shares',
       'abs_title_sentiment_polarity', 'kw_min_avg'],
      dtype='object')
```



In [196]:
```
1  sortList = [['kw_avg_max', 'LDA_00', 'min_negative_polarity', 'LDA_02', 'LDA_04',
2         'title_subjectivity', 'num_imgs', 'data_channel', 'global_subjectivity',
3         'LDA_01', 'max_negative_polarity', 'neg_pos_avg', 'LDA_03',
4         'avg_negative_polarity', 'is_weekend', 'avg_len',
5         'min_positive_polarity', 'neg_pos_min', 'avg_positive_polarity',
6         'global_rate_positive_words', 'global_sentiment_polarity',
7         'self_reference_avg_sharess', 'title_sentiment_polarity', 'shares',
8         'average_token_length', 'num_keywords', 'rate_positive_words',
9         'kw_min_min', 'self_reference_max_shares', 'kw_avg_min', 'weekdays',
10        'n_unique_tokens', 'num_hrefs', 'max_positive_polarity',
11        'n_tokens_title', 'kw_avg_avg', 'kw_max_min', 'n_tokens_content',
12        'global_rate_negative_words', 'num_videos', 'kw_min_max',
13        'num_self_hrefs', 'kw_max_avg', 'media', 'self_reference_min_shares',
14        'abs_title_sentiment_polarity', 'kw_min_avg']]
```

In [197]:
```
1 sortList.remove('shares')
```

In [198]:
```
1 #Top 10 Features
```

In [199]:
```
1 list10 = sortList[-10:]
2 list10.append('shares')
```

In [259]:
```
1 polynomial(2,dataset[list10])
```

```
MAE : 0.05092958952889084
MSE : 0.004583979286133878
RMSE : 0.06770509054815507
r2_score : 0.09020490058504604  9.0
```

In [304]:
```
1 polynomial(3,dataset[list10])
```

```
MAE : 0.05096046053492994
MSE : 0.004713327825636697
RMSE : 0.06865368035026743
r2_score : 0.06453273672686288  6.0
```

In [200]:
```
#Top 15 Features
```

In [201]:
```
list15 = sortList[-15:]
list15.append('shares')
```

In [264]:
```
polynomial(2,dataset[list15])
```

```
MAE : 0.05017333544136189
MSE : 0.004502672679140369
RMSE : 0.06710195734209524
r2_score : 0.10634204867742769  11.0
```

In [4]:
```
#Top 25 Features
```

In [202]:
```
list25 = sortList[-25:]
list25.append('shares')
```

In [203]:
```
polynomial(2,dataset[list25])
```

```
Polynomial Model.....................................

MAE : 0.04961281960568475
MSE : 0.004542502695658321
RMSE : 0.06739809118705307
r2_score : 0.09843687468432871  10.0
```

In [5]:
```
#Top 35 Features
```

In [209]:
```
list35 = sortList[-35:]
list35.append('shares')
```

In [270]:
```
polynomial(2,dataset[list35])
```

```
MAE : 0.049183918013428876
MSE : 0.004494553019660496
RMSE : 0.06704142763739818
r2_score : 0.10795358004408362  11.0
```

In [6]:
```
#Top 45 Features
```

In [204]:
```
list45 = sortList[-45:]
list45.append('shares')
```

In [307]:
```
polynomial(2,dataset[list35])
```

```
MAE : 0.049183918013428876
MSE : 0.004494553019660496
RMSE : 0.06704142763739818
r2_score : 0.10795358004408362  11.0
```

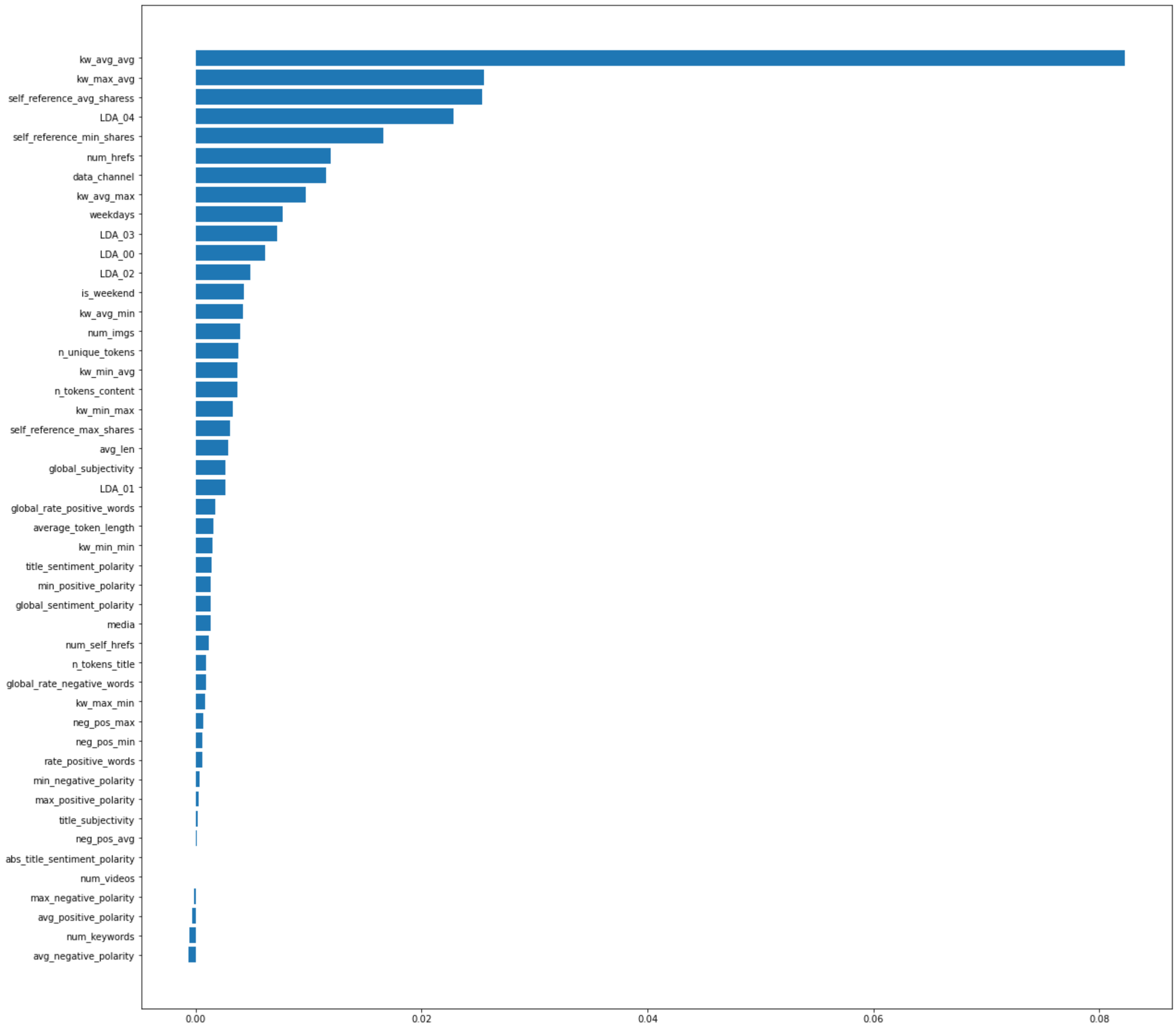
2.Permutation Based Feature

The permutation based importance can be used to overcome drawbacks of default feature importance computed with mean impurity decrease.

In [207]:
```
def Permutation(dataset,n):
    X = dataset[dataset.columns.difference(['shares'])]
    Y = dataset['shares']
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
    rf = RandomForestRegressor(n_estimators=n)
    rf.fit(X_train, Y_train)
    perm_importance = permutation_importance(rf, X_test, Y_test)
    plt.figure(figsize=(20,20))
    sorted_idx = perm_importance.importances_mean.argsort()
    plt.barh(X_train.columns[sorted_idx], perm_importance.importances_mean[sorted_idx])
    return dataset.columns[sorted_idx]
```

In [208]:
```
1 Permutation(dataset,100)
```

Out[208]:
```
Index(['num_keywords', 'min_negative_polarity', 'weekdays', 'LDA_01',
       'title_subjectivity', 'num_imgs', 'rate_positive_words', 'neg_pos_avg',
       'LDA_02', 'LDA_04', 'title_sentiment_polarity', 'min_positive_polarity',
       'avg_positive_polarity', 'self_reference_max_shares', 'kw_min_min',
       'global_rate_positive_words', 'max_negative_polarity', 'LDA_03',
       'kw_avg_min', 'global_subjectivity', 'avg_len', 'LDA_00', 'num_videos',
       'kw_max_min', 'n_tokens_content', 'kw_min_max', 'average_token_length',
       'shares', 'is_weekend', 'global_sentiment_polarity',
       'self_reference_avg_sharess', 'global_rate_negative_words',
       'avg_negative_polarity', 'kw_avg_avg', 'kw_avg_max', 'n_unique_tokens',
       'n_tokens_title', 'num_hrefs', 'neg_pos_min', 'kw_max_avg',
       'data_channel', 'max_positive_polarity', 'media', 'num_self_hrefs',
       'abs_title_sentiment_polarity', 'self_reference_min_shares',
       'kw_min_avg'],
      dtype='object')
```



In [ ]:
```
1 SHAP(dataset)
```

Finding better way

how about using other models

In [212]:
```
1 def regressionTest(dataset):
2     simpleLinear(dataset)
3     ridge_regression(dataset,10)
4     lasso_regression(dataset, 10)
5     randomforestReg(dataset)
6     gradientBoostingRegressor(dataset)
7     svr(dataset)
8
```

In [214]:
```
1 #With all features
```

In [213]:
```
regressionTest(dataset)
```

```
Linear Model.....................................

MAE : 0.049787095087776435
MSE : 0.004447808465286461
RMSE : 0.06669189205058185
r2_score : 0.11723110156833583  12.0

Ridge Model.....................................

The train score for ridge model is 0.10695438696269477
The test score for ridge model is 0.10963325071325125
MAE : 0.050119141400555164
MSE : 0.004486090042051646
r2_score : 0.10963325071325125  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.....................................

MAE : 0.048544176757464634
MSE : 0.004252350004827501
r2_score : 0.15602428503726307  16.0

GradientBoostingRegresso.....................................

MAE : 0.04791658386715039
MSE : 0.00420438479680897
r2_score : 0.16554407308030272  17.0

SVR.....................................

MAE : 0.05760787773364369
MSE : 0.004993109524176087
r2_score : 0.00900368601603141  1.0
```

In [215]:
```
#Top 10 Features
```

In [216]:
```
regressionTest(dataset[list10])
```

```
Linear Model.....................................

MAE : 0.05253123087890712
MSE : 0.004776853804539528
RMSE : 0.06911478716265809
r2_score : 0.0519245592714892  5.0

Ridge Model.....................................

The train score for ridge model is 0.050242263110322205
The test score for ridge model is 0.05182085047046958
MAE : 0.05253547755002729
MSE : 0.004777376338674927
r2_score : 0.05182085047046958  5.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.....................................

MAE : 0.05055626783453161
MSE : 0.004524306409747984
r2_score : 0.10204834208315716  10.0

GradientBoostingRegresso.....................................

MAE : 0.0496789794601733
MSE : 0.0044039697077798996
r2_score : 0.12593190151840472  13.0

SVR.....................................

MAE : 0.05780666908960624
MSE : 0.005073521112181063
r2_score : -0.006955825172074936 4  -1.0
```

In [217]:
```
#Top 15 Features
```

In [218]:
```
1 regressionTest(dataset[list15])
```

```
Linear Model.....................................

MAE : 0.051394030109551445
MSE : 0.004622392325527062
RMSE : 0.06798817783649641
r2_score : 0.0825809579770882  8.0

Ridge Model.....................................

The train score for ridge model is 0.07383381209459172
The test score for ridge model is 0.0772731983554037
MAE : 0.05161519010031025
MSE : 0.004649135336318421
r2_score : 0.0772731983554037  8.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.............................

MAE : 0.04973942626360882
MSE : 0.004407862339898601
r2_score : 0.1251593200112996  13.0

GradientBoostingRegresso..........................

MAE : 0.04920614905853955
MSE : 0.0043427729268268375
r2_score : 0.13807779658810604  14.000000000000002

SVR...............................................

MAE : 0.0576424962590647
MSE : 0.005026330682907967
r2_score : 0.0024101903816647896  0.0
```

In [219]:
```
1 #Top 25 Features
```

In [220]:
```
1 regressionTest(dataset[list25])
```

```
Linear Model.....................................

MAE : 0.05067749932135206
MSE : 0.004534396307203992
RMSE : 0.06733792621698408
r2_score : 0.10004577211368937  10.0

Ridge Model.....................................

The train score for ridge model is 0.08780212767154905
The test score for ridge model is 0.09431770395819339
MAE : 0.05094980923242152
MSE : 0.004563257031768502
r2_score : 0.09431770395819339  9.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.............................

MAE : 0.049194162777240855
MSE : 0.004343027020489626
r2_score : 0.13802736591319076  14.000000000000002

GradientBoostingRegresso..........................

MAE : 0.048568218458628465
MSE : 0.004265237354582877
r2_score : 0.15346649694093495  15.0

SVR...............................................

MAE : 0.05796224464351116
MSE : 0.005044907999990108
r2_score : -0.0012769013320204081  -0.0
```

In [221]:
```
1 #Top 35 Features
```

In [222]:
```
1 regressionTest(dataset[list35])
```

```
Linear Model.....................................

MAE : 0.050278965530543424
MSE : 0.0044888656026475845
RMSE : 0.06699899702717635
r2_score : 0.10908237749356786  11.0

Ridge Model.....................................

The train score for ridge model is 0.09719379463649125
The test score for ridge model is 0.10136011885468932
MAE : 0.050613999604374585
MSE : 0.004527774004842266
r2_score : 0.10136011885468932  10.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.....................................

MAE : 0.04905994730029079
MSE : 0.004334132604608011
r2_score : 0.13979266533453172  14.000000000000002

GradientBoostingRegresso.....................................

MAE : 0.04835284124676989
MSE : 0.004256209333102124
r2_score : 0.15525831343658258  16.0

SVR.....................................

MAE : 0.05806035770586252
MSE : 0.005056952269511673
r2_score : -0.003667360952966048  -0.0
```

In [223]:
```
1 #Top 45 Features
```

In [224]:
```
1 regressionTest(dataset[list45])
```

```
Linear Model.....................................

MAE : 0.04978660156768577
MSE : 0.004447746024865052
RMSE : 0.0666914239229082
r2_score : 0.11724349429218062  12.0

Ridge Model.....................................

The train score for ridge model is 0.10695164977203242
The test score for ridge model is 0.10963657419566597
MAE : 0.05011920642381809
MSE : 0.004486073296770695
r2_score : 0.10963657419566597  11.0

Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.00024812584822586636
MAE : 0.05409789631295404
MSE : 0.005039724541087296
r2_score : -0.00024812584822586636  -0.0

RandomForestregressor.....................................

MAE : 0.04861678568360429
MSE : 0.004269306981712732
r2_score : 0.15265878674242073  15.0

GradientBoostingRegresso.....................................

MAE : 0.04790927753813888
MSE : 0.004197005247061614
r2_score : 0.1670087128129344  17.0

SVR.....................................

MAE : 0.05765211583462231
MSE : 0.004998763477654005
r2_score : 0.007881530167271444  1.0
```

Previous datasets

1.OnlineNewsPopularity

- two non predictive features removed

In [315]:
```
1 regressionTest(OnlineNewsPopularity)
```

```
Linear Model.....................................

MAE : 60745495.04133717
MSE : 2.925520993211126e+19
RMSE : 5408808550.144039
r2_score : -384409419567.89185  -38440941956789.0

Ridge Model.....................................

The train score for ridge model is 0.021939678798331474
The test score for ridge model is -0.3358153134312516
MAE : 3085.732251457612
MSE : 101661289.85285343
r2_score : -0.3358153134312516  -34.0

Lasso Model.....................................

The train score for ls model is 0.02129886307728668
The test score for ls model is 0.024839810969102483
MAE : 3022.1138232510402
MSE : 74213883.93533753
r2_score : 0.024839810969102483  2.0

RandomForestregressor...........................

MAE : 3404.460279984866
MSE : 82785420.88325486
r2_score : -0.08778900115046318  -9.0

GradientBoostingRegresso........................

MAE : 3034.9506606331247
MSE : 79249255.68025947
r2_score : -0.04132427858181309  -4.0

SVR.............................................

MAE : 2403.541838327482
MSE : 79747989.35506949
r2_score : -0.04787757021424621  -5.0
```

2.Dataset_withContetn

- Before deleting records with No content

In [316]:
```
1 regressionTest(dataset_withContetn)
```

```
Linear Model.....................................

MAE : 241.14547330187565
MSE : 458621856.91970253
RMSE : 21415.458363521022
r2_score : -534333453.04510975  -53433345305.0

Ridge Model.....................................

The train score for ridge model is 0.12816800456595312
The test score for ridge model is -0.07899986223920963
MAE : 0.6482271165314156
MSE : 0.9261125551657352
r2_score : -0.07899986223920963  -8.0

Lasso Model.....................................

The train score for ls model is 0.08370922882846588
The test score for ls model is 0.07482549346511824
MAE : 0.6687128272738687
MSE : 0.7940832582157135
r2_score : 0.07482549346511824  7.000000000000001

RandomForestregressor...........................

MAE : 0.6400404890649998
MSE : 0.7381292793992626
r2_score : 0.14001663583536894  14.000000000000002

GradientBoostingRegresso........................

MAE : 0.6287198324416955
MSE : 0.7230011289972897
r2_score : 0.15764221720624405  16.0

SVR.............................................

MAE : 0.6625194851740547
MSE : 0.8452359904581144
r2_score : 0.015228211542045234  2.0
```

3.Dataset_Yhat

- Before unifying the categorical features

In [317]:
```
1 regressionTest(dataset_Yhat)
```

```
Linear Model.....................................

MAE : 0.6361076401522979
MSE : 0.7365150240927806
RMSE : 0.858204535115482
r2_score : 0.10971945662174454  11.0

Ridge Model.....................................

The train score for ridge model is 0.133441885859363
The test score for ridge model is 0.10977360536822334
MAE : 0.6359250264231455
MSE : 0.7364702276905519
r2_score : 0.10977360536822334  11.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001

RandomForestregressor...........................

MAE : 0.633272955053526
MSE : 0.7206997460516168
r2_score : 0.12883656064213966  13.0

GradientBoostingRegresso........................

MAE : 0.6219619277752815
MSE : 0.7061963175582089
r2_score : 0.14636793444658258  15.0

SVR.............................................

MAE : 0.6576152581813225
MSE : 0.8273264637282131
r2_score : -5.109140364090159e-05  -0.0
```

4.Dataset_Multicollinearity

- Before deleting highly correlated features

In [318]:
```
1 regressionTest(dataset_Multicollinearity)
```

```
Linear Model.....................................

MAE : 0.6405895162738783
MSE : 0.7416601385526579
RMSE : 0.8611969220524757
r2_score : 0.10350017371882592  10.0

Ridge Model.....................................

The train score for ridge model is 0.12314944020578034
The test score for ridge model is 0.10341063719023047
MAE : 0.6406756249817277
MSE : 0.7417342107077849
r2_score : 0.10341063719023047  10.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001

RandomForestregressor...........................

MAE : 0.6346815491836639
MSE : 0.7221881589667448
r2_score : 0.12703740513885076  13.0

GradientBoostingRegresso........................

MAE : 0.6237828388745705
MSE : 0.707957815458972
r2_score : 0.14423868078990032  14.000000000000002

SVR.............................................

MAE : 0.6575885758841706
MSE : 0.8272685086253097
r2_score : 1.8963244012737412e-05  0.0
```

5.Dataset_RAW

- Before analyzing the dataset and changing distributions

In [319]:
```
1 regressionTest(dataset_RAW)
```

```
Linear Model.....................................

MAE : 0.6406746917318741
MSE : 0.7416572118386511
RMSE : 0.8611952228378018
r2_score : 0.10350371145594062  10.0

Ridge Model.....................................

The train score for ridge model is 0.12280559453490081
The test score for ridge model is 0.10345919339787413
MAE : 0.6407935748647433
MSE : 0.7416940409245536
r2_score : 0.10345919339787413  10.0

Lasso Model.....................................

The train score for ls model is 0.08638325687973347
The test score for ls model is 0.07190115273751874
MAE : 0.65851506373399
MSE : 0.7678015092390749
r2_score : 0.07190115273751874  7.000000000000001

RandomForestregressor...........................

MAE : 0.6337689977623829
MSE : 0.7209308753307782
r2_score : 0.12855717747477524  13.0

GradientBoostingRegresso........................

MAE : 0.6232221486701339
MSE : 0.7074808306786852
r2_score : 0.14481524780548827  14.000000000000002

SVR.............................................

MAE : 0.6575813386092544
MSE : 0.827205072724451
r2_score : 9.564294020170117e-05  0.0
```

6.Datset_Outlier

- Before outlier handling

In [320]:
```
1 regressionTest(datset_Outlier)
```

```
Linear Model.....................................

MAE : 0.6542459913255024
MSE : 0.7712790702335444
RMSE : 0.8782249542307167
r2_score : 0.11219872817225007  11.0

Ridge Model.....................................

The train score for ridge model is 0.12245159308102938
The test score for ridge model is 0.11208314576499856
MAE : 0.6541825940496352
MSE : 0.7713794826731625
r2_score : 0.11208314576499856  11.0

Lasso Model.....................................

The train score for ls model is 0.08339856811972257
The test score for ls model is 0.07684970875771802
MAE : 0.6727242559460294
MSE : 0.8019885991482513
r2_score : 0.07684970875771802  8.0

RandomForestregressor...........................

MAE : 0.6438859297193837
MSE : 0.7459386546168482
r2_score : 0.14136748703191215  14.000000000000002

GradientBoostingRegresso........................

MAE : 0.633405296076723
MSE : 0.7309836897853732
r2_score : 0.15858179675446504  16.0

SVR.............................................

MAE : 0.6569785808871642
MSE : 0.8349272371147515
r2_score : 0.03893481412680588  4.0
```

7.Dataset_featureScale

- Before feature scaling

In [321]:
```
1 regressionTest(dataset_featureScale)
```

```
Linear Model.....................................

MAE : 0.6448283721337132
MSE : 0.7461141579054319
RMSE : 0.8637789983007412
r2_score : 0.11725124633813078  12.0

Ridge Model.....................................

The train score for ridge model is 0.11410783663418123
The test score for ridge model is 0.1172178566392219
MAE : 0.6449151479226611
MSE : 0.7461423794429654
r2_score : 0.1172178566392219  12.0

Lasso Model.....................................

The train score for ls model is 0.022033483754844152
The test score for ls model is 0.01783201880637053
MAE : 0.6916114347031985
MSE : 0.8301449683956852
r2_score : 0.01783201880637053  2.0

RandomForestregressor.....................................

MAE : 0.627900932569232
MSE : 0.712990074771809
r2_score : 0.15644128554672054  16.0

GradientBoostingRegresso.....................................

MAE : 0.6200681820758455
MSE : 0.7039285520084094
r2_score : 0.16716222930702307  17.0

SVR.....................................................

MAE : 0.6480117298457555
MSE : 0.8172501274384777
r2_score : 0.03308826969376588  3.0
```

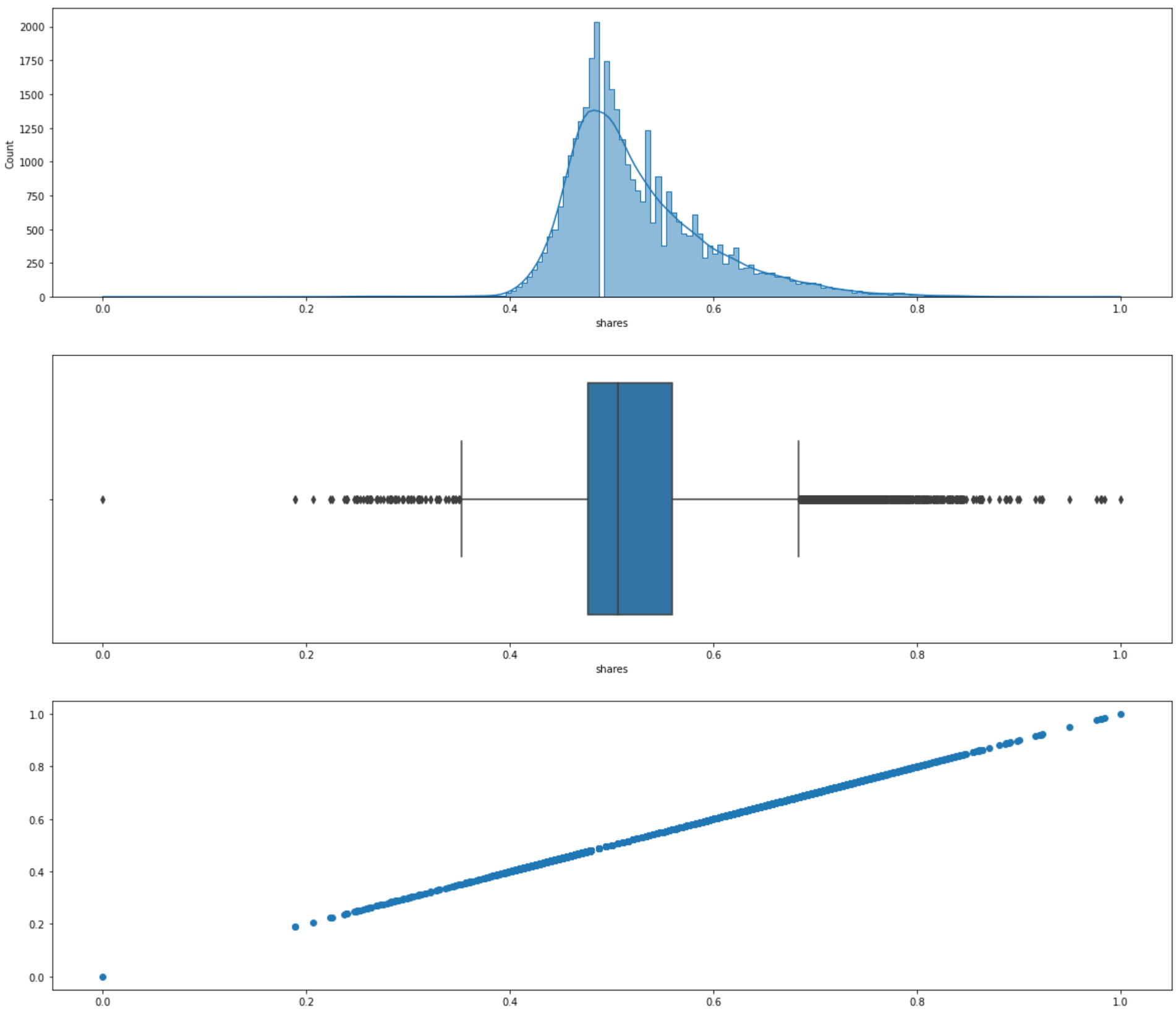
Getting low accuracy in all regression models so let's convert it to classification

using classification models implemented in models part Models

In [322]:
```
1 featureAnalysis('shares',dataset)
```

Out[322]:

	count	mean	std	min	25%	50%	75%	max
shares	34293.0	0.524973	0.070981	0.0	0.476508	0.505855	0.559344	1.0



first with equal size

In [195]:
```
def testClassification_Qcut(datasetClass):
    dataset_class = datasetClass.copy()
    dataset_class['target'] = pd.qcut(datasetClass['shares'], q=5,
                                      labels=['very_low', 'low', 'medium', 'high', 'very_high'])
    dataset_class.drop('shares', inplace=True, axis=1)
    decisionTree(dataset_class)
    randomForestClassifier(dataset_class)
    knn(dataset_class)
```

In [196]:
```
def testClassification_Qcut3(datasetClass):
    dataset_class = datasetClass.copy()
    dataset_class['target'] = pd.qcut(datasetClass['shares'], q=3,
                                      labels=['low', 'medium', 'high'])
    dataset_class.drop('shares', inplace=True, axis=1)
    decisionTree(dataset_class)
    randomForestClassifier(dataset_class)
    knn(dataset_class)
```

1.OnlineNewsPopularity

- two non predictive features removed

In [197]:
```
testClassification_Qcut(OnlineNewsPopularity)
```

```
DecisionTree.....................................
Accuracy: 0.25198637911464244
              precision    recall  f1-score   support

    very_low       0.23      0.22      0.22      1453
      medium       0.25      0.26      0.25      1789
         low       0.20      0.21      0.21      1505
   very_high       0.27      0.27      0.27      1560
        high       0.30      0.30      0.30      1622

    accuracy                           0.25      7929
   macro avg       0.25      0.25      0.25      7929
weighted avg       0.25      0.25      0.25      7929


RandomForest.....................................

Accuracy: 0.32299155000630597
              precision    recall  f1-score   support

    very_low       0.29      0.22      0.25      1453
      medium       0.27      0.30      0.28      1789
         low       0.26      0.16      0.19      1505
   very_high       0.35      0.47      0.40      1560
        high       0.39      0.45      0.42      1622

    accuracy                           0.32      7929
   macro avg       0.31      0.32      0.31      7929
weighted avg       0.31      0.32      0.31      7929


KNN.............................................

Accuracy: 0.23975280615462227
              precision    recall  f1-score   support

    very_low       0.20      0.27      0.23      1453
      medium       0.25      0.30      0.28      1789
         low       0.20      0.19      0.20      1505
   very_high       0.28      0.21      0.24      1560
        high       0.29      0.21      0.25      1622

    accuracy                           0.24      7929
   macro avg       0.24      0.24      0.24      7929
weighted avg       0.25      0.24      0.24      7929
```

In [198]:
```
testClassification_Qcut3(OnlineNewsPopularity)
```

```
DecisionTree.....................................

Accuracy: 0.40068104426787743
              precision    recall  f1-score   support

         low       0.41      0.40      0.40      2571
      medium       0.46      0.46      0.46      2986
        high       0.32      0.32      0.32      2372

    accuracy                           0.40      7929
   macro avg       0.40      0.40      0.40      7929
weighted avg       0.40      0.40      0.40      7929


RandomForest.....................................

Accuracy: 0.5111615588346576
              precision    recall  f1-score   support

         low       0.50      0.58      0.53      2571
      medium       0.56      0.69      0.62      2986
        high       0.41      0.21      0.28      2372

    accuracy                           0.51      7929
   macro avg       0.49      0.49      0.48      7929
weighted avg       0.49      0.51      0.49      7929


KNN.............................................

Accuracy: 0.4035817883717997
              precision    recall  f1-score   support

         low       0.39      0.50      0.44      2571
      medium       0.46      0.49      0.47      2986
        high       0.31      0.19      0.23      2372

    accuracy                           0.40      7929
   macro avg       0.39      0.39      0.38      7929
weighted avg       0.39      0.40      0.39      7929
```

2.Dataset_withContetn

- Before deleting records with No content

In [199]:
```
1 testClassification_Qcut(dataset_Yhat)
```

```
DecisionTree.....................................

Accuracy: 0.23956843884050436
              precision    recall  f1-score   support

    very_low       0.20      0.19      0.19      1467
      medium       0.24      0.25      0.25      1723
         low       0.19      0.20      0.20      1493
   very_high       0.27      0.26      0.27      1490
        high       0.29      0.29      0.29      1520

    accuracy                           0.24      7693
   macro avg       0.24      0.24      0.24      7693
weighted avg       0.24      0.24      0.24      7693


RandomForest.....................................

Accuracy: 0.3215910568048876
              precision    recall  f1-score   support

    very_low       0.29      0.21      0.24      1467
      medium       0.29      0.34      0.31      1723
         low       0.24      0.14      0.18      1493
   very_high       0.36      0.47      0.41      1490
        high       0.38      0.44      0.41      1520

    accuracy                           0.32      7693
   macro avg       0.31      0.32      0.31      7693
weighted avg       0.31      0.32      0.31      7693


KNN.............................................

Accuracy: 0.23475887170154686
              precision    recall  f1-score   support

    very_low       0.20      0.25      0.22      1467
      medium       0.26      0.32      0.29      1723
         low       0.19      0.18      0.19      1493
   very_high       0.27      0.21      0.23      1490
        high       0.27      0.20      0.23      1520

    accuracy                           0.23      7693
   macro avg       0.24      0.23      0.23      7693
weighted avg       0.24      0.23      0.23      7693
```

In [200]:
```
1 testClassification_Qcut3(dataset_Yhat)
```

```
DecisionTree.....................................

Accuracy: 0.3960743533082023
              precision    recall  f1-score   support

         low       0.40      0.39      0.39      2530
      medium       0.46      0.46      0.46      2843
        high       0.32      0.32      0.32      2320

    accuracy                           0.40      7693
   macro avg       0.39      0.39      0.39      7693
weighted avg       0.40      0.40      0.40      7693


RandomForest.....................................

Accuracy: 0.49980501754842066
              precision    recall  f1-score   support

         low       0.50      0.57      0.53      2530
      medium       0.53      0.68      0.60      2843
        high       0.40      0.20      0.27      2320

    accuracy                           0.50      7693
   macro avg       0.48      0.48      0.47      7693
weighted avg       0.48      0.50      0.48      7693


KNN.............................................

Accuracy: 0.3976342129208371
              precision    recall  f1-score   support

         low       0.39      0.47      0.43      2530
      medium       0.44      0.50      0.47      2843
        high       0.32      0.19      0.24      2320

    accuracy                           0.40      7693
   macro avg       0.38      0.39      0.38      7693
weighted avg       0.39      0.40      0.39      7693
```

3.Datset_Outlier

- Before outlier handling

In [201]:
```
testClassification_Qcut(datset_Outlier)
```

```
DecisionTree.....................................

Accuracy: 0.24658780709736125
              precision    recall  f1-score   support

    very_low       0.21      0.22      0.21      1410
      medium       0.24      0.23      0.24      1721
         low       0.20      0.21      0.20      1462
   very_high       0.28      0.29      0.29      1509
        high       0.29      0.28      0.29      1591

    accuracy                           0.25      7693
   macro avg       0.25      0.25      0.25      7693
weighted avg       0.25      0.25      0.25      7693


RandomForest.....................................

Accuracy: 0.32042116209541144
              precision    recall  f1-score   support

    very_low       0.29      0.23      0.25      1410
      medium       0.28      0.33      0.30      1721
         low       0.24      0.14      0.18      1462
   very_high       0.36      0.47      0.41      1509
        high       0.38      0.41      0.40      1591

    accuracy                           0.32      7693
   macro avg       0.31      0.32      0.31      7693
weighted avg       0.31      0.32      0.31      7693


KNN..............................................

Accuracy: 0.2283894449499545
              precision    recall  f1-score   support

    very_low       0.19      0.24      0.21      1410
      medium       0.23      0.29      0.26      1721
         low       0.20      0.19      0.19      1462
   very_high       0.27      0.21      0.24      1509
        high       0.28      0.20      0.23      1591

    accuracy                           0.23      7693
   macro avg       0.23      0.23      0.23      7693
weighted avg       0.23      0.23      0.23      7693
```

In [202]:
```
testClassification_Qcut3(datset_Outlier)
```

```
DecisionTree.....................................

Accuracy: 0.4012738853503185
              precision    recall  f1-score   support

         low       0.42      0.42      0.42      2494
      medium       0.47      0.45      0.46      2904
        high       0.31      0.32      0.32      2295

    accuracy                           0.40      7693
   macro avg       0.40      0.40      0.40      7693
weighted avg       0.40      0.40      0.40      7693


RandomForest.....................................

Accuracy: 0.5042246197842194
              precision    recall  f1-score   support

         low       0.50      0.58      0.54      2494
      medium       0.55      0.68      0.61      2904
        high       0.38      0.19      0.26      2295

    accuracy                           0.50      7693
   macro avg       0.48      0.49      0.47      7693
weighted avg       0.48      0.50      0.48      7693


KNN..............................................

Accuracy: 0.40335369816716493
              precision    recall  f1-score   support

         low       0.40      0.49      0.44      2494
      medium       0.45      0.50      0.48      2904
        high       0.30      0.18      0.23      2295

    accuracy                           0.40      7693
   macro avg       0.38      0.39      0.38      7693
weighted avg       0.39      0.40      0.39      7693
```

4.Datset

- Final result

In [203]:
```
1 testClassification_Qcut(dataset)
```

```
DecisionTree.....................................

Accuracy: 0.2426009622393935
              precision    recall  f1-score   support

    very_low       0.22      0.23      0.23      1246
      medium       0.24      0.24      0.24      1509
         low       0.20      0.21      0.20      1354
   very_high       0.27      0.26      0.27      1358
        high       0.28      0.27      0.28      1392

    accuracy                           0.24      6859
   macro avg       0.24      0.24      0.24      6859
weighted avg       0.24      0.24      0.24      6859


RandomForest.....................................

Accuracy: 0.33386791077416533
              precision    recall  f1-score   support

    very_low       0.29      0.23      0.26      1246
      medium       0.28      0.29      0.28      1509
         low       0.30      0.16      0.21      1354
   very_high       0.36      0.51      0.42      1358
        high       0.39      0.48      0.43      1392

    accuracy                           0.33      6859
   macro avg       0.32      0.33      0.32      6859
weighted avg       0.32      0.33      0.32      6859


KNN.............................................

Accuracy: 0.2565971715993585
              precision    recall  f1-score   support

    very_low       0.22      0.27      0.24      1246
      medium       0.26      0.32      0.29      1509
         low       0.21      0.19      0.20      1354
   very_high       0.29      0.22      0.25      1358
        high       0.32      0.27      0.30      1392

    accuracy                           0.26      6859
   macro avg       0.26      0.26      0.25      6859
weighted avg       0.26      0.26      0.26      6859
```

In [204]:
```
1 testClassification_Qcut3(dataset)
```

```
DecisionTree.....................................

Accuracy: 0.41186761918647036
              precision    recall  f1-score   support

         low       0.40      0.41      0.41      2220
      medium       0.49      0.48      0.48      2549
        high       0.33      0.33      0.33      2090

    accuracy                           0.41      6859
   macro avg       0.41      0.41      0.41      6859
weighted avg       0.41      0.41      0.41      6859


RandomForest.....................................

Accuracy: 0.5082373523837294
              precision    recall  f1-score   support

         low       0.50      0.61      0.55      2220
      medium       0.55      0.68      0.61      2549
        high       0.41      0.19      0.26      2090

    accuracy                           0.51      6859
   macro avg       0.49      0.49      0.47      6859
weighted avg       0.49      0.51      0.48      6859


KNN.............................................

Accuracy: 0.44642076104388395
              precision    recall  f1-score   support

         low       0.44      0.52      0.48      2220
      medium       0.49      0.58      0.53      2549
        high       0.35      0.20      0.26      2090

    accuracy                           0.45      6859
   macro avg       0.43      0.43      0.42      6859
weighted avg       0.43      0.45      0.43      6859
```

Exercise 13

Implement batch gradient descent with early stopping for softmax regression from scratch. Use it on a classification task on the Penguins dataset.

In [205]:
```
1 df = sns.load_dataset('penguins')
```

In [206]:
```
df
```

79	Adelie	Torgersen	42.1	19.1	195.0	4000.0	Male
80	Adelie	Torgersen	34.6	17.2	189.0	3200.0	Female
81	Adelie	Torgersen	42.9	17.6	196.0	4700.0	Male
82	Adelie	Torgersen	36.7	18.8	187.0	3800.0	Female
83	Adelie	Torgersen	35.1	19.4	193.0	4200.0	Male
84	Adelie	Dream	37.3	17.8	191.0	3350.0	Female
85	Adelie	Dream	41.3	20.3	194.0	3550.0	Male
86	Adelie	Dream	36.3	19.5	190.0	3800.0	Male
87	Adelie	Dream	36.9	18.6	189.0	3500.0	Female
88	Adelie	Dream	38.3	19.2	189.0	3950.0	Male
89	Adelie	Dream	38.9	18.8	190.0	3600.0	Female
90	Adelie	Dream	35.7	18.0	202.0	3550.0	Female
91	Adelie	Dream	41.1	18.1	205.0	4300.0	Male

In [207]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

In [208]:
```
#we can drop null values according to number of them
df = df.dropna()
```

In [209]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 333 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            333 non-null    object 
 1   island             333 non-null    object 
 2   bill_length_mm     333 non-null    float64
 3   bill_depth_mm      333 non-null    float64
 4   flipper_length_mm  333 non-null    float64
 5   body_mass_g        333 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 20.8+ KB
```

In [210]:
```
df['gender'] = 0
df.loc[df['sex'] == 'Male', 'gender'] = 1
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [211]:
```
df.drop('sex', inplace=True, axis=1)
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [212]:
```
#encode categorical features
df = pd.get_dummies(df, columns=['species', 'island'])
```

In [213]:
```
df
```

64	36.4	17.1	184.0	2850.0	0	1	0	0	1	0	0
65	41.6	18.0	192.0	3950.0	1	1	0	0	1	0	0
66	35.5	16.2	195.0	3350.0	0	1	0	0	1	0	0
67	41.1	19.1	188.0	4100.0	1	1	0	0	1	0	0
68	35.9	16.6	190.0	3050.0	0	1	0	0	0	0	1
69	41.8	19.4	198.0	4450.0	1	1	0	0	0	0	1
70	33.5	19.0	190.0	3600.0	0	1	0	0	0	0	1
71	39.7	18.4	190.0	3900.0	1	1	0	0	0	0	1
72	39.6	17.2	196.0	3550.0	0	1	0	0	0	0	1
73	45.8	18.9	197.0	4150.0	1	1	0	0	0	0	1
74	35.5	17.5	190.0	3700.0	0	1	0	0	0	0	1
75	42.8	18.5	195.0	4250.0	1	1	0	0	0	0	1
76	40.9	16.8	191.0	3700.0	0	1	0	0	0	0	1

In [214]:
```
df.shape
```

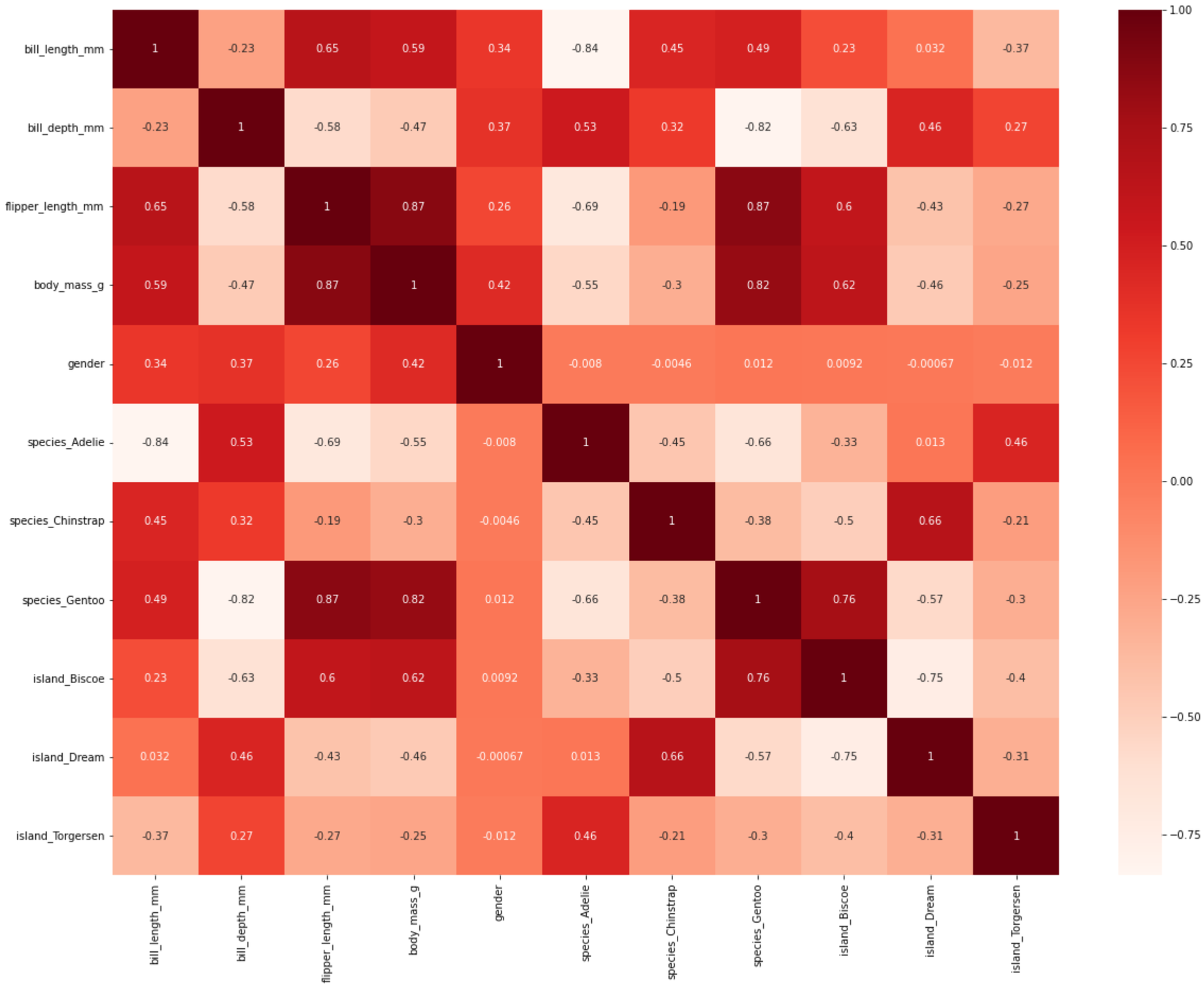
Out[214]: (333, 11)

In [215]:
```
df.describe()
```

Out[215]:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	gender	species_Adelie	species_Chinstrap	species_Gentoo	island_Biscoe	island_Dream	island_Torgersen
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000
mean	43.992793	17.164865	200.966967	4207.057057	0.504505	0.438438	0.204204	0.357357	0.489489	0.369369	0.141141
std	5.468668	1.969235	14.015765	805.215802	0.500732	0.496942	0.403726	0.479942	0.500642	0.483360	0.348691
min	32.100000	13.100000	172.000000	2700.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	39.500000	15.600000	190.000000	3550.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	44.500000	17.300000	197.000000	4050.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	48.600000	18.700000	213.000000	4775.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000
max	59.600000	21.500000	231.000000	6300.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [216]:
```
plt.figure(figsize=(20,15))
cor = df.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



It looks like there is no correlation between species and island with the target(obviously)

In [217]:
```
df.columns
```

Out[217]:
```
Index(['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g',
       'gender', 'species_Adelie', 'species_Chinstrap', 'species_Gentoo',
       'island_Biscoe', 'island_Dream', 'island_Torgersen'],
      dtype='object')
```

In [218]:
```
#dropping them
df.drop('species_Adelie', inplace=True, axis=1)
df.drop('species_Chinstrap', inplace=True, axis=1)
df.drop('species_Gentoo', inplace=True, axis=1)
df.drop('island_Biscoe', inplace=True, axis=1)
df.drop('island_Dream', inplace=True, axis=1)
df.drop('island_Torgersen', inplace=True, axis=1)
```

In [219]:
```
df
```

60	35.7	16.9	185.0	3150.0	0
61	41.3	21.1	195.0	4400.0	1
62	37.6	17.0	185.0	3600.0	0
63	41.1	18.2	192.0	4050.0	1
64	36.4	17.1	184.0	2850.0	0
65	41.6	18.0	192.0	3950.0	1
66	35.5	16.2	195.0	3350.0	0
67	41.1	19.1	188.0	4100.0	1
68	35.9	16.6	190.0	3050.0	0
69	41.8	19.4	198.0	4450.0	1
70	33.5	19.0	190.0	3600.0	0
71	39.7	18.4	190.0	3900.0	1
72	39.6	17.2	196.0	3550.0	0

scaling numerical values

In [220]:
```
#first we should split dataset
X = df[df.columns.difference(['gender'])]
Y = df['gender']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/5, random_state = 0 )
```

In [221]:
```
scaler = MinMaxScaler()
scale = MinMaxScaler().fit(X_train)
X_train = scale.transform(X_train)
```

softmax function

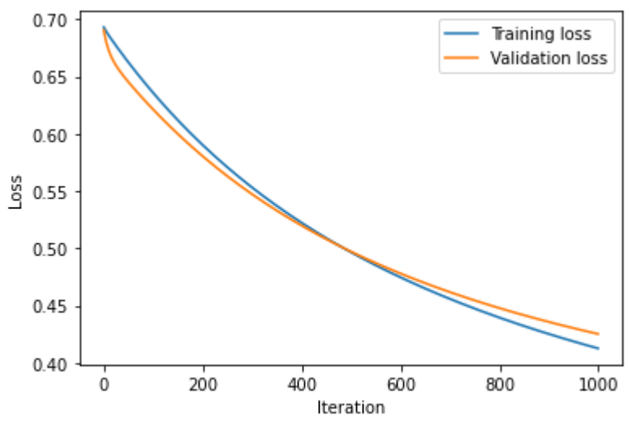
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

In [239]:
```
def softmax(x):
    #for the error
    x -= np.max(x, axis=1, keepdims=True)
    return np.exp(x) / np.sum(np.exp(x), axis=1, keepdims=True)
```

In [240]:
```
def cross_entropy_loss(y_pred, y_true):
    m = y_true.shape[0]
    log_likelihood = -np.log(y_pred[range(m), y_true])
    loss = np.sum(log_likelihood) / m
    return loss
```

In [241]:
```
def batch_gradient_descent(X, y, learning_rate=0.1, max_iters=1000, tol=1e-4, early_stopping_rounds=10, validation_size=0.2):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=validation_size, random_state=42)


    n_features = X_train.shape[1]
    n_classes = len(np.unique(y_train))
    W = np.zeros((n_features, n_classes))
    b = np.zeros(n_classes)


    train_losses = []
    val_losses = []

    patience = early_stopping_rounds
    best_val_loss = np.inf

    # Perform batch gradient descent
    for i in range(max_iters):

        y_pred_train = softmax(np.dot(X_train, W) + b)
        grad_w = np.dot(X_train.T, (y_pred_train - pd.get_dummies(y_train).values)) / X_train.shape[0]
        grad_b = np.mean(y_pred_train - pd.get_dummies(y_train).values, axis=0)

        # Update the weights and biases
        W -= learning_rate * grad_w
        b -= learning_rate * grad_b


        train_loss = cross_entropy_loss(y_pred_train, y_train)
        val_loss = cross_entropy_loss(softmax(np.dot(X_val, W) + b), y_val)


        train_losses.append(train_loss)
        val_losses.append(val_loss)


        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience = early_stopping_rounds
        else:
            patience -= 1
            if patience == 0:
                print(f'Early stopping after {i} iterations.')
            break

        if train_loss < tol:
            print(f'Tolerance level reached after {i} iterations.')
            break
    plt.plot(train_losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    return W, b
```

In [245]:
```
W, b = batch_gradient_descent(X_train, Y_train)

y_pred_test = softmax(np.dot(X_test, W) + b)
y_pred_labels = np.argmax(y_pred_test, axis=1)

# Compute the accuracy and the classification report
accuracy = np.mean(y_pred_labels == Y_test)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(Y_test, y_pred_labels, zero_division=1))

```




```
Accuracy: 0.39
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        41
           1       0.39      1.00      0.56        26

    accuracy                           0.39        67
   macro avg       0.69      0.50      0.28        67
weighted avg       0.76      0.39      0.22        67
```